

---

Bachelorarbeit Medientechnologie

# **Effizientes und realistisches Partikelsystem zur Simulation von Feuer und Rauch in VR- Umgebung**

vorgelegt von

**Miro Steiger**

Erstgutachter: Prof. Dr.-Ing. Arnulph Fuhrmann (Technische Hochschule Köln)

Zweitgutachter: Prof. Dr. rer. nat. Stefan Michael Grünvogel (Technische Hochschule Köln)

Köln, 28.07.2022

Fakultät für  
Informations-, Medien-  
und Elektrotechnik

**Technology  
Arts Sciences  
TH Köln**

# Bachelorarbeit

**Titel:** Effizientes und realistisches Partikelsystem zur Simulation von Feuer und Rauch in VR-Umgebung

**Gutachter:**

- Prof. Dr. Arnulph Fuhrmann (TH Köln)
- Prof. Dr. rer. nat. Stefan Michael Grünvogel (TH Köln)

**Zusammenfassung:** Der Einsatz von Virtual Reality findet in immer mehr Bereichen seinen Nutzen. Im Bereich der Brandbekämpfung könnte die Technik eine sichere und kostengünstigere Alternative zu bestehenden Trainingsmethoden sein. Aktuelle Anwendungen legen bisher jedoch nicht viel Wert auf wirklich immersive Erfahrungen beim Rendering der Brände. Durch realistischere Renderings kann der Nutzer in echte Stresssituationen versetzt werden. Parallax Occlusion Mapping und Ray Marching sind zwei Methoden, welche sich für die Darstellung von Feuer und Rauch in virtueller Realität anbieten könnten. In dieser Arbeit werden daher die beiden vorgeschlagenen Algorithmen in einem Partikelsystem implementiert, verglichen und hinsichtlich ihrer Performance, optischer Qualität und den Einsatzmöglichkeiten in einem Brandsimulator bewertet.

**Stichwörter:** Virtual Reality, Partikelsystem, Parallax Occlusion Mapping, Volumen Rendering, Ray Marching, Echtzeitrendering

**Datum:** 28.07.2022

# Bachelors Thesis

**Title:** Efficient and realistic particle system to render fire and smoke in VR

**Reviewers:**

- Prof. Dr. Arnulph Fuhrmann (TH Köln)
- Prof. Dr. rer. nat. Stefan Michael Grünvogel (TH Köln)

**Abstract:** The use of virtual reality extends into more and more areas. In context of firefighting, the technique could be a safe and cheaper alternative to existing fire training methods. However, current applications focused less on truly immersive experiences when it comes to the rendering of the fires. More realistic renderings can put the user in higher stressful situations. Parallax occlusion mapping and ray marching are two methods that could be suitable for displaying fire and smoke in a stereoscopic view. In this thesis, the two proposed algorithms are implemented in a particle system, compared and evaluated with regard to their performance, appearance and possible uses in a fire simulator.

**Keywords:** Virtual Reality, Particle System, Parallax Occlusion Mapping, Volume Rendering, Ray Marching, Real Time Rendering

**Date:** 28.07.2022

# Inhalt

<b>1 Einleitung</b>	<b>1</b>
1.1 Problem . . . . .	1
1.2 Zielsetzung . . . . .	2
1.3 Struktur der Arbeit . . . . .	2
<b>2 Related Work</b>	<b>2</b>
2.1 Partikelsysteme . . . . .	2
2.2 Parallax Occlusion Mapping . . . . .	3
2.3 Ray Marching . . . . .	3
<b>3 Grundlagen</b>	<b>4</b>
3.1 Virtual Reality . . . . .	4
3.1.1 Konzept . . . . .	4
3.1.2 Tiefenwahrnehmung . . . . .	5
3.1.3 Head-Mounted-Display . . . . .	5
3.2 Feuer- und Rauchsimulationen . . . . .	6
3.2.1 Eigenschaften . . . . .	6
3.2.2 Partikelsysteme . . . . .	7
3.3 Texture Mapping . . . . .	8
3.3.1 Bump Mapping . . . . .	8
3.3.2 Displacement Mapping . . . . .	9
3.3.3 Parallax Mapping . . . . .	10
3.3.4 Parallax Occlusion Mapping . . . . .	10
3.4 Volume Rendering . . . . .	11
3.4.1 Ray Marching . . . . .	12
3.4.2 Volume Ray Marching . . . . .	13
<b>4 Konzeption und Umsetzung</b>	<b>15</b>
4.1 Idee . . . . .	15
4.2 Erstellung der Texturen . . . . .	16
4.2.1 Texture Sheets . . . . .	16
4.2.2 Volume Textures . . . . .	18
4.3 Shader . . . . .	19
4.3.1 Parallax Occlusion Mapping . . . . .	19
4.3.2 Raymarcher . . . . .	20
4.4 Partikelsysteme . . . . .	21
<b>5 Evaluierung</b>	<b>22</b>
5.1 Parallax Occlusion Mapping . . . . .	22
5.2 Ray Marching . . . . .	25
<b>6 Schluss</b>	<b>27</b>
6.1 Fazit . . . . .	27

6.2 Limitationen . . . . .	27
6.3 Ausblick . . . . .	28
<b>Literaturverzeichnis</b>	<b>29</b>

# 1 Einleitung

Die Simulation von Feuer und Rauch ist ein viel diskutiertes Thema in der Computergrafik. Einerseits spielen diese besonders in Videospielen und Filmproduktionen eine außerordentlich große Rolle, auf der anderen Seite helfen solche Simulationen auch im Bereich der Gefahrenbekämpfung und -vorbeugung. So kann eine realistische Feuersimulation dazu beitragen, einen Trainingssimulator für die Brandbekämpfung [Schlager, 2017] zu entwerfen. Der Nutzer kann dabei mithilfe von Head-Mounted-Displays in ein virtuelles Brandszenario versetzt werden, ohne dabei wirklichen Gefahren ausgesetzt zu sein. Eine solche Anwendung findet seinen Nutzen sowohl im Training von Einsatzkräften, als auch bereits bei der Entscheidung, einem solchen Beruf nachzugehen. Hierbei gibt es verschiedenste Ansätze, um ein künstlich erzeugtes Feuer auf einem Bildschirm anzeigen zu lassen. Eine gängige Methode für das Rendering von Gasen und Flüssigkeiten in Videospielen, in denen sich auch Feuer und Rauch aufgrund ihrer physikalischen Eigenschaften wiederfinden, sind der Einsatz von Partikelsystemen. Die physikalisch korrekte Simulation kann dabei, unter anderem auf Basis von Fluidsimulationen, sehr realitätsnah dargestellt werden. Auf realen, physikalischen Eigenschaften basierende Simulationen sind jedoch sehr aufwändig in der Berechnung und bisher kaum für die Echtzeitanwendung gedacht. Gerade in Virtual-Reality-Systemen, in denen die Performance besonders wichtig für das Nutzererlebnis sind, eignet sich die aufwändige Simulation von Fluiden aufgrund ihrer Performance nicht. Als Alternative hat sich hierfür eine Art von Partikelsystem etabliert, welche sich anstatt der physikalisch korrekten Eigenschaften eher an einer optischen Illusion unter Anwendung animierter Texturen bedienen. Hierbei ist das Konzept des 'Billboardings' ein weit verbreiteter und beliebter Ansatz, um realistischere Renderings der Partikel zu erzeugen. Diese bieten eine optisch überzeugende und dabei noch effiziente Lösung.

## 1.1 Problem

Ein solches Partikelsystem, basierend auf Texturen, kann auf einem flachen Bildschirm realistisch und optisch überzeugend aussehen. In einer VR-Umgebung gerät diese Methode jedoch leider an seine Grenzen. Die Illusion basiert auf der Eigenschaft, dass die Normalen der flachen Partikel immer zum Betrachter, bzw. der Kamera orientiert sind. Dadurch lässt sich nicht erkennen, dass lediglich flache Texturen zum Einsatz kommen. Aufgrund dieser Eigenschaft sehen die Partikel voluminös aus und täuschen Tiefe vor. In VR-Anwendungen müssen jedoch immer zwei Bilder erzeugt werden, eins für jedes Auge. Dadurch, dass sich die flachen Billboards immer zur jeweiligen Kamera, bzw. zum Mittelpunkt zwischen beiden Augen orientieren, zerstört dies die Illusion, und es lässt sich erkennen, dass die Partikel ihre Tiefe lediglich vortäuschen. So sieht ein Feuer-Partikelsystem, basierend auf Texturen, schnell sehr unecht aus und die Immersion ist gestört. Um in einem Trainingsszenario der Feuerwehr jedoch einen wirklichen Nutzen zu finden, sollte das Feuer so realistisch wie möglich aussehen. Erst dann wird der Nutzer in eine echte Stresssituation versetzt und kann sich somit besser auf einen Einsatz in der realen Welt vorbereiten.

## 1.2 Zielsetzung

Ziel dieser Arbeit ist es, zwei Methoden für das Rendering von Feuer und Rauch zu implementieren und zu bewerten, inwiefern sich damit in Virtual Reality-Umgebungen ein realistisches Bild dieser Phänomene erzeugen lassen kann. Durch eine bessere, plausible Darstellung in der Stereo-Ansicht kann beim Nutzer ein reales Gefühl von Gefahr hervorgerufen werden, welches den Trainingseffekt deutlich erhöhen kann. Dazu sollen beide Varianten in Unity entworfen werden, sodass überprüft werden kann, ob die Methoden sowohl in der Stereoansicht funktionieren, als auch in Hinblick auf die benötigte Performance für VR-Renderings die Mindest-Framerate einhalten können.

## 1.3 Struktur der Arbeit

Zunächst wird in Kapitel 2: **Related Work** ein kurzer Überblick über bereits vorhandene, verwandte Arbeiten und Beiträge aus den Bereichen Partikelsystemen, Parallax Occlusion Mapping, Ray Marching und deren Anwendung in VR gegeben. Das Kapitel 3: **Grundlagen** setzt sich mit den thematischen Grundlagen zu VR, Feuer- und Rauchsimulationen, Mapping Verfahren und Volume Rendering auseinander, welche benötigt werden, um die zu vergleichenden Methoden zu implementieren. Im vierten Teil der Arbeit geht es um die **Konzeption und Umsetzung** der gewählten Methoden. Hierbei werden die Schritte von der Idee über die Erstellung der Texturen und Shader bis hin zur Anwendung in Partikelsystemen beschrieben. Anschließend werden im 5. Kapitel **Evaluierung** beide Methoden individuell auf ihre Vor- und Nachteile bewertet und hinsichtlich der Performance und ihrer optischen Qualität und Nutzbarkeit in VR diskutiert. Im letzten Teil werden in Kapitel 6: **Schluss** eine kurze Zusammenfassung der Ergebnisse, sowie die Limitationen bei der Durchführung der Arbeit beschrieben. Es wird schlussendlich ein Überblick über weitergehende Möglichkeiten gegeben, die über den Umfang dieser Arbeit hinausgehen.

# 2 Related Work

Die Idee eines Einsatztrainings für die Brandbekämpfung in Virtual Reality ist nicht neu. Es gibt mit 'Serious Games' sogar eine eigene Kategorie, bei der es sich um Videospiele handelt, bei denen der Bildungsaspekt im Vordergrund steht. Auch in der Brandvorbeugung und -bekämpfung entstanden bereits einige Anwendungen, die versuchen, sich die Möglichkeiten von VR zunutze zu machen. Das Ziel der Anwendungen ist dabei oft dasselbe. Sowohl um die Einsatzkräfte in realistischen Szenarios zu trainieren, ohne dabei die physische Gesundheit der Personen aufs Spiel zu setzen [Williams-Bell et al., 2014], als auch die Umwelt und die finanziellen Mittel zu schonen.

## 2.1 Partikelsysteme

Für die Simulation von Feuer und Rauch werden in computergenerierten Welten aufgrund ihrer Performance seit vielen Jahren überwiegend Partikelsysteme benutzt. Partikelsysteme sind eine kostengünstige Lösung in Hinblick auf die Rechenzeit. Daher eignen sich diese besonders um

solche volumetrischen Effekte, die schwer zu modellieren und zu berechnen sind, in Echtzeit-systemen darzustellen zu können. Dies erzeugt auf flachen Bildschirmen die Illusion, dass diese Texturen keine flachen Bilder, sondern voluminös sind. Hierbei handelt es sich um Systeme von einzelnen Partikeln, welche über verschiedene Eigenschaften verfügen und diverse Formen annehmen können. Ein solches System kann Partikel in Form von beispielsweise Punkten, Linien, Sprites oder Meshes emittieren [Reeves, 1983].

In Schlager [2017] wurde bereits unter Anwendung von Partikelsystemen ein interaktiver Trainingssimulator für die Anwendung eines Feuerlöschers entwickelt. Der Nutzer lernt den korrekten Umgang mit einem Feuerlöscher und muss für die gegebenen Situationen aus verschiedenen Löschmitteln das jeweils am besten geeignete Mittel auswählen und den Brand löschen. Solche Trainings gibt es zwar bereits mit echtem Brand und Feuerlöschern, jedoch lernt der Nutzer hierbei nichts darüber, wie sich ein Feuer in Innenräumen verhält. Auch die Rauchentwicklung im Raum wird hier nicht weiter betrachtet. Der Fokus der Arbeit liegt hier auf der approximiert-realistischen Simulation der Ausbreitung des Feuers, basierend auf Daten des Fire Dynamics Simulators (FDS) [McGrattan und Forney, 2004]. FDS ist eine auf Computational Fluid Dynamics (CFD) basierende Open-Source Software vom National Institute of Standards and Technology (NIST). Brennbare Objekte werden durch ein Voxelgitter repräsentiert, in dem jedes Voxel Informationen wie Temperatur und Brennbarkeit beinhaltet. Bei Schlager liegt der Fokus nicht auf dem realistischen Rendering. In Hinblick auf die Performance kam Schlager zu dem Fazit, dass man einen Kompromiss zwischen realistischem Rendering und realitätsnahem Verhalten des Feuers finden muss.

## 2.2 Parallax Occlusion Mapping

Parallax Occlusion Mapping (POM) [Brawley und Tatarchuk, 2004] wurde als Weiterentwicklung des Parallax Mappings [Kaneko et al., 2001] präsentiert. POM wird mittlerweile in vielen Bereichen aufgrund der höheren Details bei gering aufgelösten Meshes [Tatarchuk, 2006] als Alternative zu Vertexdisplacements, bzw. hochauflösenden Geometrien verwendet. Es wurde mit dem Prism Parallax Occlusion Mapping eine Methode vorgestellt, um die Silhouetten der Oberflächen korrekt darstellen zu können [Dachsbacher und Tatarchuk, 2007]. Außerdem gibt es bereits Versuche, POM im Zusammenhang mit Fluidsimulationen anzuwenden [Kufner, 2017]. Hierbei wurde eine 2D-Fluidsimulation, unter anderem mithilfe von POM, in eine Art Pseudo-Volumen überführt. Der von Kufner festgestellte Nachteil seiner Implementierung ist ein sichtbarer Treppeneffekt. Da eine höhere Samplerate – die diesem Treppeneffekt entgegenkommen würde – bedeutet, dass die Performance negativ beeinflusst wird, ist der Nutzen von POM für diese Art der Simulation eingeschränkt.

## 2.3 Ray Marching

Partikelsysteme, basierend auf Billboards, haben das Problem der Darstellung in VR. Alle Vorteile, die diese Art von Partikelsystem mit sich bringt, gehen durch stereoskopisches Rendering verloren, da die Partikel plötzlich flach aussehen oder sich zusammen mit der Bewegung des VR-Headset drehen und kippen können. Eine Methode, um volumetrische Effekte zu rendern, ist das Ray Marching. Ray Marching im Zusammenhang mit SDFs gibt es schon seit über 30

Jahren [J. C. Hart et al., 1989]. Dieser Ansatz bietet neben deutlich realistischeren Renderings den Vorteil, dass dieses auch in der Stereoansicht gut funktioniert und überzeugende Ergebnisse liefert [Wald et al., 2006]. Die Methode bietet jedoch den Nachteil einer aufwändigeren Berechnung, und daher auch längeren Renderingzeiten. Es wurde außerdem bereits von Zhang [2020] versucht, die Charakteristiken von volumetrischen Effekten auf ein Billboard-System anzuwenden. Hierbei hängt die Real Time-Performance allerdings stark von der Komplexität der zu rendernden Szene ab.

## 3 Grundlagen

Dieses Kapitel umfasst die technischen Grundlagen, auf denen diese Arbeit aufbaut. Es wird ein Einblick in die Funktionsweise von VR gegeben, um das Problem bisheriger Lösungen besser zu verstehen. Außerdem werden bewährte Methoden zur Feuer- und Rauchsimulation gesammelt und erklärt. Ein weiterer Teil dieses Kapitels setzt sich mit grundlegendem Texture Mapping, Bump- und Displacement Mapping, sowie Parallax- und Parallax Occlusion Mapping auseinander. Abschließend werden noch zwei Volume Rendering-Methoden vorgestellt: Ray Marching und Texturbasiertes Volume Rendering.

### 3.1 Virtual Reality

#### 3.1.1 Konzept

Hinter dem Begriff Virtual Reality (VR) verbirgt sich das Konzept einer künstlichen, von Computern generierten Welt. Der Nutzer kann in diese Welt eintauchen und hat dabei die Möglichkeit, sich als Betrachter in dieser Welt umzuschauen, oder sogar mit dieser Welt zu interagieren. Das erste Konzept eines VR-Headsets mit Kopftracking wurde bereits in den 60er Jahren von Ivan Sutherland entworfen. [Sutherland, 1965, 1968]

Heutzutage gibt es verschiedene Arten von VR. Zum einen die "Non-immersive Virtual Reality". Hierbei steuert der Nutzer seine virtuelle Umgebung, ist sich dabei aber noch bewusst, in welcher Realität er sich tatsächlich befindet. Die Interaktion geschieht üblicherweise durch Eingabegeräte wie Controller, Maus oder Tastatur. Ein weit verbreitetes Anwendungsgebiet sind dabei herkömmliche Videospiele. Gegenüberstehend gibt es dagegen die "Fully Immersive Virtual Reality". Hierbei wird der Nutzer durch spezielle Hardware, zum Beispiel mithilfe eines Head-Mounted-Displays (HMD), einem sogenannten VR-Headset, selbst in eine virtuelle dreidimensionale Umgebung versetzt. Durch visuelles, auditives und teilweise auch haptisches Feedback kann der Nutzer dabei immer weiter in die virtuelle Welt eintauchen. Auch hier verfügt der Nutzer über spezielle Eingabegeräte wie dem Headset, Controllern oder Laufbändern. Diese sind jedoch in ihrer Benutzung näher an der bekannten Realität. So kann sich der Nutzer z.B. mit einer Kopfbewegung in der virtuellen Welt umsehen oder Dinge anfassen und mit diesen interagieren. Dieser Einfluss auf die Umgebung sorgt dafür, dass sich eine Simulation echter anfühlen kann.

Die Idee von Fully Immersive Virtual Realities baut dabei darauf auf, die Sinne des Nutzers so überzeugend zu täuschen, sodass dieser glaubt, er befände sich in einer anderen Welt. Die nächste

Stufe nach Immersion ist die Präsenz. Präsenz beschreibt hierbei das Gefühl, bzw. die Illusion, dass sich der Nutzer tatsächlich physisch in dieser computergenerierten Welt befindet und diese nicht mehr von seiner wirklichen Realität unterscheiden kann [Schuemie et al., 2001].

### 3.1.2 Tiefenwahrnehmung

Der Mensch ist ein visuell orientiertes Lebewesen. Daher ist der Einsatz einer VR-Brille einer der wichtigsten Faktoren, um eine solche Illusion zu erzeugen. Der Eindruck, sich in einer anderen dreidimensionalen Welt zu befinden, wird von der Illusion von Raum und Tiefe vom Gehirn erzeugt. Die Information dazu werden aus den Bildern beider Augen generiert. Das Gehirn hat einige Möglichkeiten, sich ein Verständnis von Räumlichkeit zu schaffen. Die Tiefenwahrnehmung wird aufgrund binokularer Disparität erzeugt. Dieser Begriff bezeichnet grob gesagt den kleinen, aber bedeutenden Unterschied zwischen den beiden einzelnen Bildern, welche von den Augen erzeugt werden. Daraus kann das Gehirn in etwa abschätzen, wie weit ein Objekt entfernt ist. Diese Disparitäten entstehen durch Informationen wie Verdeckung, Schattenwurf oder die unterschiedlichen Orientierungen von Linien zwischen beiden Blickwinkeln [Tauer, 2010]. Mit Hilfe aller dieser Informationen entsteht ein Eindruck von räumlicher Tiefe, jedoch ist es noch nicht ganz möglich, mit den Methoden eine wirklich genaue Einschätzung der Entfernung in dieser Welt zu erhalten. Gerade transparente Objekte lassen sich nicht genau verorten [El Jamiy und Marsh, 2019].

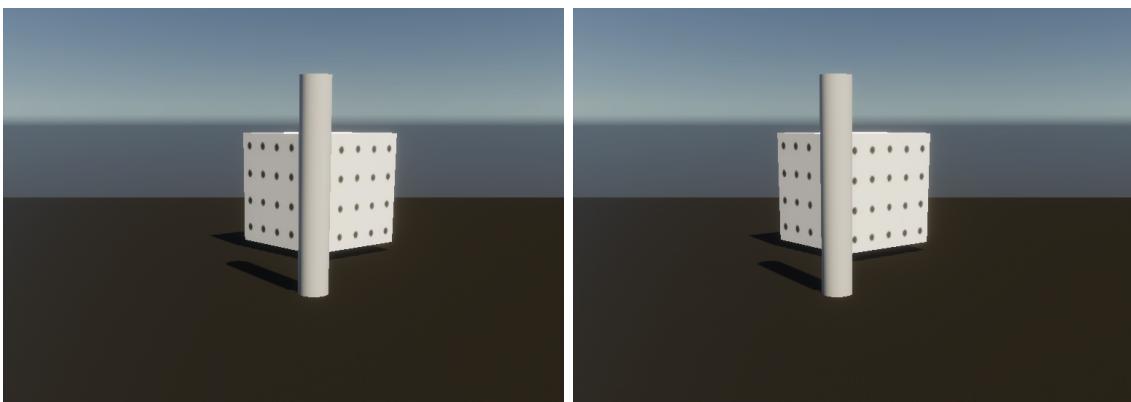


Abbildung 1: Perspektive des linken, sowie des rechten Auges. Die Anzahl der sichtbaren Punkte auf den Seiten des Würfels verdeutlicht die leicht verschiedenen Blickwinkel der Augen.

### 3.1.3 Head-Mounted-Display

Die meisten kommerziellen Systeme basieren heutzutage auf der Nutzung eines HMD. Diese können sowohl Bild, als auch Ton ausgeben. Für diese Arbeit ist jedoch nur die visuelle Komponente interessant. Ein HMD basiert auf zwei Displays, welche sich direkt vor den Augen des Nutzers befinden [Sutherland, 1968]. Diese Displays machen sich die Eigenschaften des menschlichen Sehens zunutze, um die Illusion von Tiefe hervorzurufen. Auf jedem Display wird jeweils ein Bild gerendert, welches aus leicht verschobenen Positionen heraus berechnet wird. Dabei werden in der Software, anstatt der üblichen einzelnen Kamera für das Rendering, die Bilder von zwei virtuellen Kameras aufgenommen, welche die Abstände der beiden Augen simulieren [Gateau und Nash, 2010]. Dieser Abstand beträgt den durchschnittlichen Augenabstand eines Menschen. Normalerweise liegt dieser bei ca. 65mm.

Der Einsatz einer VR-Brille bringt einige technische Anforderungen mit sich, welche sich deutlich von denen eines herkömmlichen Monitors unterscheiden. Die empfohlenen Spezifikationen unterscheiden sich dabei je nach Art und Hersteller der Brille. Für die Implementierung und das Testing der Methoden dieser Arbeit wurde die HP Reverb G2 mit den folgenden Spezifikationen verwendet:

Bildschirm	2 x 2,89-Zoll-LCD
Auflösung	2160 x 2160 pro Auge 4320 x 2160 kombiniert
Field OF View	~114°
Bildrate	90Hz
Trackingarchitektur	6DoF
Augenabstand	64 mm +/- 4 mm durch Hardware Slider

Tabelle 1: HP Reverb G2 Headset Spezifikationen [HP Development Company, 2022]

## 3.2 Feuer- und Rauchsimulationen

### 3.2.1 Eigenschaften

Chemische Reaktionen wie Feuer, oder der aus winzigen Partikeln bestehende Rauch, lassen sich nur schwer physikalisch exakt simulieren. Es haben sich hierfür verschiedene Ansätze entwickelt, um ein annähernd realistisches Rendering dieser Phänomene (mit volumetrischem Charakter) zu erschaffen. Es gibt hierbei beispielsweise die physikalisch basierten Ansätze. Diese beruhen auf aufwändig berechneten Fluidsimulationen und werden vor allem in der Gefahrensimulation verwendet. Diese sind aufgrund ihrer Komplexität aber nur bedingt echtzeitfähig und werden meist zuvor berechnet. Der Fokus liegt dabei meist auf der realistischen Ausbreitung des Feuers und des Rauchs, um beispielsweise Fluchtwege zu testen oder Evakuierungsszenarien zu simulieren. Diese Anwendungsfälle müssen nicht in Echtzeit berechnet werden. Daher ist diese Art der Simulation auch für Anwendungen, in denen es um die Interaktion mit dem Feuer geht, nicht effizient nutzbar. Ein Beispiel dafür ist der Einsatz in einem Simulator, der den Umgang mit dem Feuerlöscher näher bringen soll. Diese basieren zumeist auf effizienteren Partikelsystemen, mit denen sich solche Phänomene durch eine Vielzahl an Parametern ebenfalls darstellen lassen. Dieser Ansatz ist dabei eher von artistischer Natur. Partikel lassen sich zwar genau steuern, beispielsweise durch Vektorfelder. Um ein realistisches Aussehen zu erzeugen gibt es aber vor allem in Videospiele andere Methoden, die sich durchgesetzt haben.

Feuer, bzw. Flammen sind brennende, Licht und Wärme emittierende Gase, erzeugt durch eine chemische Reaktion. Feuer brennt für gewöhnlich mit Temperaturen zwischen 500°C und 1400°C, was einem Farbspektrum von dunkelrot bis hin zu weiß entspricht [Schmidt und Symes, 2011]. Hierbei kommt es allerdings immer auch auf den Stoff an, der verbrannt wird. Durch die Verbrennung werden viele, winzige Rußpartikel in die Umgebung abgegeben. Durch ihre Leich-

tigkeit und den Auftrieb der Hitze verteilen sich diese in einer Wolke nach oben. Dies drückt sich in Form einer dunklen Wolke aus. Obwohl beide Phänomene einhergehen, können diese Vorgänge als getrennte Partikelsysteme betrachtet werden. Dennoch laufen oftmals beide Volumen ineinander über. Auch können sich – aufgrund von Turbulenzen – einzelne Flammen bei größerem Feuer lösen und aufsteigen (vgl. Abbildung 2). Daher sollten die beiden Systeme für realistische Darstellungen von Bränden immer zusammen abgebildet werden.

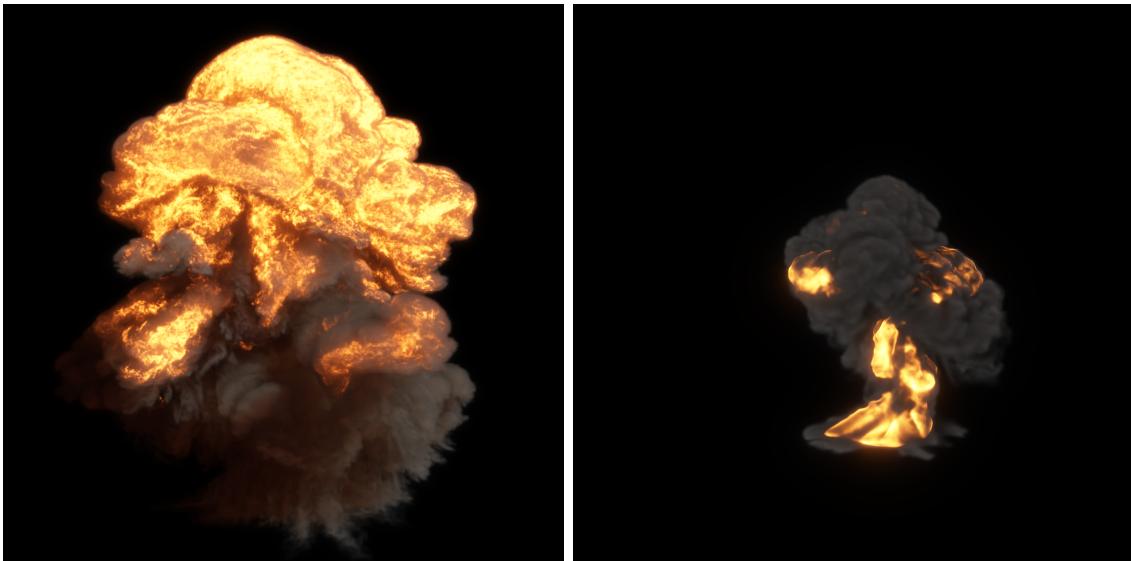


Abbildung 2: Physikalisches Rendering von Feuer. Simuliert in EmberGen von JangaFX. Links: Rendering einer Explosion. Rechts: Zu erkennen sind hier die einzelnen Flammen, die sich loslösen und nach oben steigen.

### 3.2.2 Partikelsysteme

Partikelsysteme sind Methoden, mit denen sich solche Phänomene, die normalerweise schwer, beziehungsweise gar nicht, realistisch als Geometrie darzustellen sind, simulieren. Dazu gehören neben Feuer und Rauch auch Haare, Staub und andere visuelle Effekte.

Ein Partikelsystem besteht aus einem Emitter, der viele einzelne Partikel erzeugen kann. Diese werden mit einigen Eigenschaften ausgestattet. Darunter bspw. die Lebensdauer, die Art des Partikels, die initiale Geschwindigkeit und die Richtung, in der die Partikel ausgestoßen werden oder etwa die Größe [Reeves, 1983]. Alle diese Parameter lassen sich über die Zeit, bzw. die Lebensdauer der Partikel animieren. So können die Partikel ein- und ausgeblendet werden, sie können über die Zeit auch größer, heller oder langsamer werden. Um die Partikelsimulation für ein Feuer möglichst realistisch zu machen, können die Parameter auch randomisiert werden und manipuliert werden. Partikel können auf physikalische Gegebenheiten, wie Collisionen oder Schwerkraft reagieren. Alle diese Optionen machen Partikelsysteme zu einem Werkzeug, um verschiedenste Objekte oder Effekte darzustellen, die sich durch Meshes nicht darstellen lassen könnten.

Einzelne Partikel können durch verschiedenste Arten dargestellt werden. Zu diesen Arten gehören Punkte, Linien, Sprites oder Meshes. Für Feuer und Rauch hat sich der Einsatz von Sprites, oder auch Billboards, durchgesetzt. Billboards sind simple, meistens quadratische, texturierte Polygone, die sich zum Betrachter ausrichten, sodass dieser die Sprites jederzeit von vorne sieht. Durch die

ständige Korrektur der Ausrichtung fällt nicht auf, dass die gesehenen Partikel komplett flach sind. Aus diesem Grund lassen sich volumetrische Effekte mit Billboard-basierten Partikeln, auf einen zweidimensionalen Bildschirm projiziert, gut vortäuschen. Die Texturen lassen sich zudem animieren, was die Illusion noch mehr verstärkt. Dafür werden keine Rechenressourcen bezüglich des Renderings von echten Volumen nötig, der Effekt ist jedoch vergleichbar.

### 3.3 Texture Mapping

Texture Mapping bezeichnet ein Verfahren, welches zweidimensionale Texturen auf ein dreidimensionales Objekt abbildet. Um die flachen Texturen auf die Oberfläche des Meshs abilden zu können, muss das Objekt 'UV-unwrapped' werden. Durch diesen Vorgang wird jedem Punkt auf der Oberfläche des Meshs ein Punkt auf der Textur zugewiesen [Catmull, 1974] [Blinn und Newell, 1976]. Ein sehr einfaches Beispiel zur Veranschaulichung ist dabei das Würfelnetz (vgl. Abbildung 3).

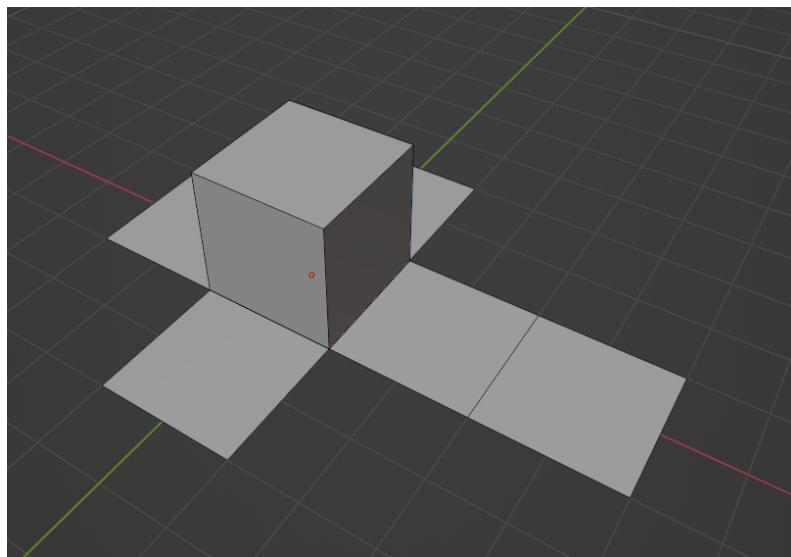


Abbildung 3: Würfelnetz.

Texture Mapping sorgt dafür, die Objekte 'anzumalen'. Reale Objekte haben oft sehr detaillierte Oberflächeneigenschaften und sind eigentlich niemals wirklich glatt. Geometrische Unebenheiten und Feinheiten, wie Kratzer, Rillen und Schmutz lassen sich zwar durch Farbtexturen andeuten, jedoch bleibt die Oberfläche komplett glatt. Diese rauen Oberflächen zu modellieren resultiert aber in einer deutlich höheren Polygon-Anzahl, was die Performance in großen Szenen schnell negativ beeinflussen kann. Daher wurden Mapping-Verfahren als Ergänzung entwickelt, um die virtuelle Auflösung solch komplexer Oberflächen kostengünstig zu erhöhen, ohne dabei die Komplexität der Geometrie zu verändern. Dabei gibt es verschiedenste Shader, welche unter Anwendung weiterer, spezieller Texturen eine deutlich detailliertere Oberfläche simulieren können.

#### 3.3.1 Bump Mapping

Eine Möglichkeit, mit der sich die Oberflächen mit mehr Details rendern lassen, ist das Rendering mithilfe von sogenannten Bump Maps. Hierbei werden auf Basis von Texturen, welche zusätz-

liche Informationen zu den Oberflächen enthalten, Details generiert, welche den Eindruck einer realistischen Struktur der Oberfläche erzeugen. Dabei ist es nicht notwendig, dass die Geometrie an sich Informationen dazu beinhaltet [Blinn, 1978].

Bump Maps sind Texturen, basierend auf Graustufen, bei denen die Helligkeit eines Pixels einen Höhenwert repräsentiert. Schwarz steht mit dem Wert 0 für den tiefsten Punkt, und weiß mit dem Wert 1 für den höchsten Punkt. Eine verbesserte Variante der Bump Maps sind die Normal Maps. Hier werden die Richtungen der Normalen in jedem Pixel durch einen Vektor repräsentiert, welcher sich aus den RGB-Werten eines jeden Pixels ergibt. Daraus lassen sich Schattierungen simulieren, welche kleine Unebenheiten (mit geringer Tiefe) wie Beulen oder Kratzer realistischer aussehen lassen. Dieser Vektor wird aus Lichtquellen, deren Einfallsrichtung und den Normalen aus der Textur berechnet. Somit wird auch bei geringer Polygonanzahl eine deutlich realistischer aussehende Oberfläche gerendert (vgl. **Abbildung 4**). [Cohen et al., 1998].

Diese Texturen bieten einen sehr kostengünstigen Ansatz, um Tiefe zu simulieren. Shaderberechnungen, basierend auf diesen Methoden, sind dabei aber nicht abhängig vom Betrachtungswinkel und sehen schnell unnatürlich verzerrt aus. Von vorne betrachtet funktioniert die Illusion. Je spitzer jedoch der Winkel zwischen Betrachter und Oberfläche wird, desto auffälliger wird die Tatsache, dass die Silhouette des Objekts immer noch flach ist, da die Geometrie hierbei nicht verändert wird. Zusätzlich wird die Effektivität von reinem Normal Mapping in VR-Umgebungen aufgrund der Unabhängigkeit vom Betrachtungswinkel geschwächt.

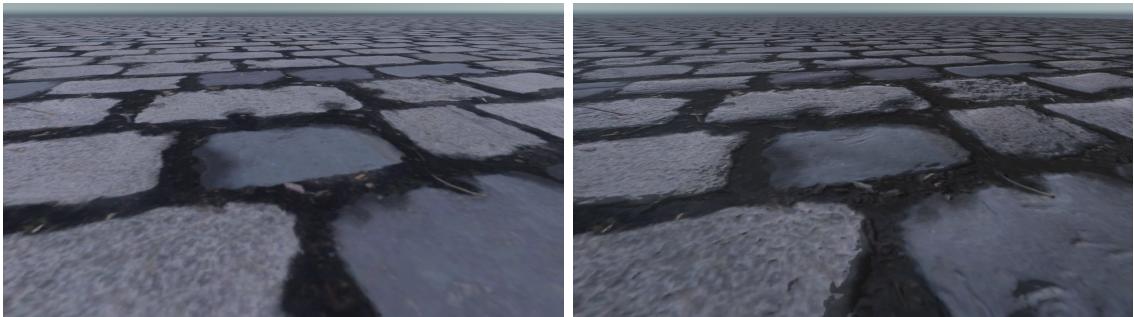


Abbildung 4: links: Einfache diffuse Beleuchtung der Textur, rechts: Normal Mapping. Beide Geometrien sind komplett flach. Hierbei sieht man gut den Unterschied zwischen dem flachen Look der diffus beleuchteten Textur und dem realistischeren Aussehen durch Einbeziehen der Oberflächennormalen aus der Normal Map.

### 3.3.2 Displacement Mapping

Mit Displacement Mapping werden dagegen tatsächlich, mithilfe von Heightmaps, die Positionen der Vertices entlang ihrer Normalen versetzt [Cook, 1984; Cook et al., 1987]. Dadurch kommt es nicht zu blickwinkelabhängigen Artefakten und die Illusion von Tiefe wird real. Objekte sehen aus einem flachen Winkel betrachtet nicht mehr glatt aus, sondern haben tatsächlich Struktur in ihrer Oberfläche. Damit dieser Effekt jedoch zustande kommt, muss das Mesh in einer gewissen Auflösung zur Verfügung stehen. Je nach Detailreichtum der Heightmap muss die Geometrie dabei in weitere Polygone unterteilt werden. Der Vorteil hierbei ist der hohe Grad an Realismus. Ein deutlicher Nachteil liegt dabei allerdings in der Performance. Ein weitgehender Einsatz von Displacement Mapping kann durch eine hohe Polygonanzahl schnell negativen Einfluss auf die Renderingzeiten nehmen.

### 3.3.3 Parallax Mapping

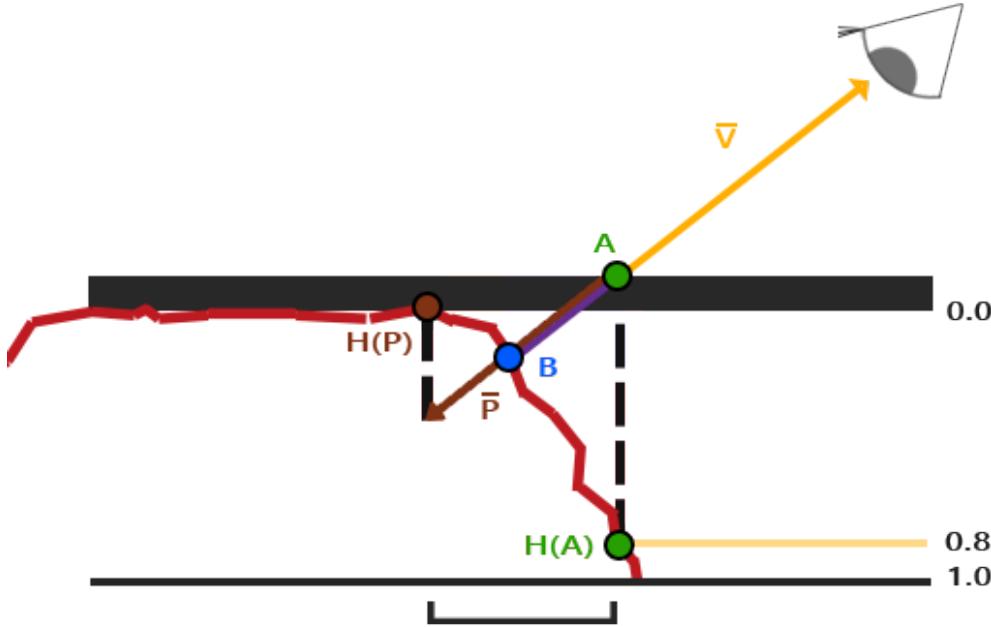


Abbildung 5: Berechnung der verschobenen Texturkoordinaten auf Basis der Heightmap. Quelle: [Vries, o.D.(a)]

Parallax Mapping (oder auch Offset (Bump)-Mapping) ist eine weitere Methode, um sich die Möglichkeiten von Bump Mapping-Verfahren zunutze zu machen. Anders als bei tatsächlicher Modifizierung der Vertices durch Displacement Maps werden hier nur die Texturkoordinaten abhängig vom Blickwinkel verschoben (vgl. Abbildung 5) [Kaneko et al., 2001; Welsh, 2004]. Durch Bewegung der Oberfläche oder des Betrachters entsteht somit ein realistischerer Eindruck von Tiefe in der Textur, welcher den des Displacement Mappings approximiert darstellt. Dabei ist Parallax Mapping allerdings immer noch deutlich effizienter als echtes Vertex-Displacement und eignet sich daher eher für Echtzeitrenderings. Parallax Mapping alleine simuliert zwar den Parallax-Effekt, jedoch ist es hiermit nicht möglich, die Silhouette zu verändern und Selbstschattierung oder -verdeckung vorzutäuschen.

### 3.3.4 Parallax Occlusion Mapping

Parallax Occlusion Mapping (POM) ist eine komplexere, verbesserte Variante des Parallax Mapping, basierend auf einem Per-Pixel Ray Marching, um die Darstellung der Oberfläche zu modifizieren. Das Ziel ist auch hier wieder das Vorherige: Detaillierte Oberflächen ohne den Preis von teuren Vertexverschiebungen. Diese Details lassen sich abhängig vom Blickwinkel aus jeder Perspektive korrekt darstellen. Durch Einsatz von dynamischer Beleuchtung und Selbstokklusion kann außerdem eine korrekte Selbstschattierung simuliert werden [Brawley und Tatarchuk, 2004; Tatarchuk, 2006]. Die Informationen aus der Heightmap werden hierbei für Berechnungen im Fragmentshader verwendet. Anstatt wie beim Displacement Mapping Details zu extrudieren, wird bei POM in die Tiefe simuliert. Hierbei wird für die Oberfläche der Wert 1.0, und damit der höchste Punkt, angenommen. Zunächst wird für jeden zu rendernden Pixel mittels Ray Castings vom Betrachter zur Geometrie-Oberfläche ein Schnittpunkt  $T_0$  ausfindig gemacht (vgl. Abbildung 6). Der Strahl  $\vec{v}$  wird nun weiter durch das – durch die Heightmap ausgedrückte –

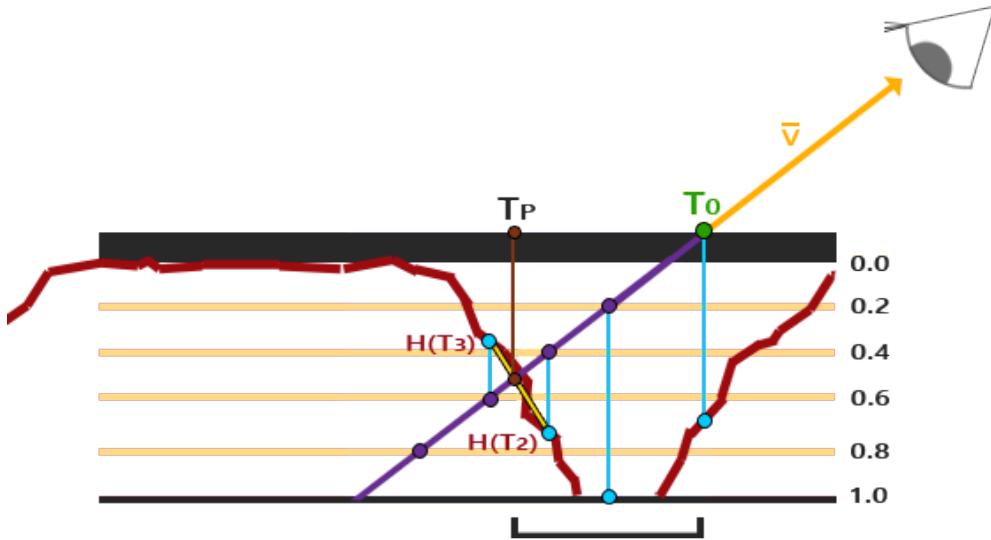


Abbildung 6: Verschiebung der Texturkoordinaten beim Parallax Occlusion Mapping. Quelle: [Vries, o.D.(b)]

Volumen verfolgt und Schrittweise abgetastet. Wenn sich der Strahl mit der Heightmap im Punkt  $T_P$  schneidet, kann mit der Position die neue, zu rendernde Texturkoordinate ermittelt werden. Liegt der Punkt  $T_P$  zwischen zwei Samplepunkten kann der tatsächliche Wert approximiert werden. Dies kann jedoch zu ungewünschten Artefakten führen. Ist die Schrittweite also zu groß, kann die Heightmap nicht detailliert genug abgetastet werden, und es entstehen Aliaseffekte. Es zeichnet sich ein Treppeneffekt ab (vgl. Abbildung 8). Generell gilt: Je geringer die Schrittweite ist, bzw. je höher die Abtastrate ist, desto genauer kann die Oberfläche gerendert werden. Jedoch erhöht sich mit einer erhöhten Sampleanzahl auch der Aufwand für die Berechnungen. Zusätzlich kann ein zu starkes Offset die Textur unnatürlich weit verzerren.

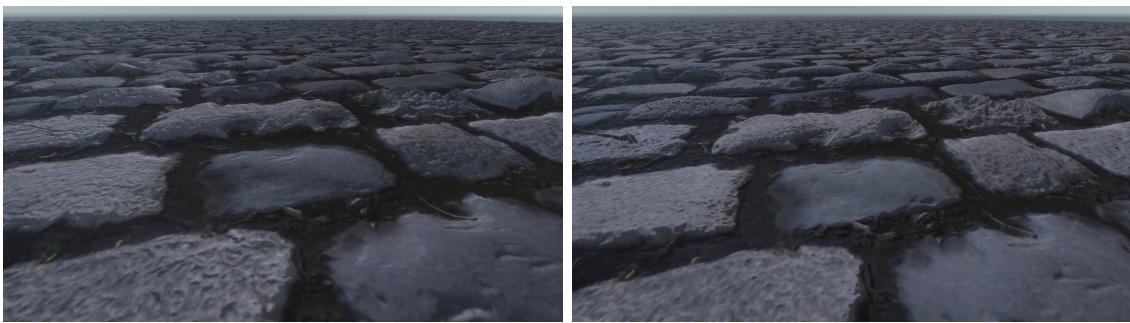


Abbildung 7: links: Displacement Map, Normal Map und 50-facher Tessellation, rechts: Parallax Occlusion Mapping mit 64 Samples. Beide Methoden sehen sich sehr ähnlich, der Unterschied ist jedoch, dass für die Displacement Map-Methode in diesem Beispiel die 50-fache Anzahl an Dreiecken gebraucht wird.

### 3.4 Volume Rendering

Unter dem Begriff Volume Rendering versteht sich eine Reihe von Methoden, die es ermöglichen, ein 3D-Datenset zu visualisieren. Anders als beim Rendern von Geometrien mit einer festen Oberfläche geht es hierbei darum, alle Daten aus dem jeweiligen Volumen darstellen zu können. Dies sind beispielsweise volumetrische Effekte wie Feuer und Rauch, Wolken oder Nebel,



Abbildung 8: Aliasingeffekt bei hohem Offset und zu geringer Sampleanzahl mit 4 Schritten.

welche sich, aufgrund ihrer gasförmigen Eigenschaften, nicht wirklich realistisch mit Geometrie darstellen lassen. Volumen haben keine wirkliche Oberfläche, sondern bestehen aus unzähligen winzigen Partikeln oder Gasen. Möchte man also solche Volumen darstellen, so geraten die bisherigen Herangehensweisen, die sich die Oberflächen der Objekte zunutze machen, schnell an ihre Grenzen. Beim Grundgedanken des Volume Renderings geht es daher um die Frage, wie das Licht durch diese Volumen wandert und welchen Einfluss die Partikel darauf haben. Es gibt einige Faktoren, die beeinflussen können, wie das Licht verändert wird, bis es das Auge nach Austritt aus einem Volumen erreicht. Während das Licht durch solch ein Volumen wandert, kann es absorbiert, reflektiert oder gestreut werden. Die Abschwächung durch Streuung und Absorption des Lichtes beim Durchlaufen eines Volumens kann vereinfacht mithilfe des Lambert-beerschen Gesetzes wie in Gleichung 3.1 approximiert werden [Mayerhöfer et al., 2020]. Der Absorptionskoeffizient  $\sigma_a$  beschreibt dabei die Wahrscheinlichkeit, dass ein Photon auf der Strecke  $d$  durch das Medium absorbiert oder anderweitig reflektiert wird.

$$I_{out} = I_{in} \cdot e^{-(d \cdot \sigma_a)}. \quad (3.1)$$

### 3.4.1 Ray Marching

Ray Marching ist eine dieser Möglichkeiten, ein Volumen darzustellen. Mittels Ray Marching lassen sich Oberflächenverschiebungen (bspw. Parallax Occlusion Mapping), dreidimensionale Volumendaten und Texturen, oder prozedural generierte Formen realisieren. Die Idee hierbei ist es – ähnlich wie beim Ray Tracing – Strahlen von der Kamera aus schrittweise durch jeden Pixel des zu rendernden Bildes zu schicken und dabei in festgelegten Abständen zu prüfen, ob der Strahl auf irgendetwas trifft. Der Unterschied besteht hierbei jedoch darin, dass sich Ray Marching auf sogenannte '*signed distance functions*' (SDF) bezieht. Mit SDFs lässt sich beschreiben, wie weit die nächstgelegene Oberfläche entfernt ist. Dazu wird die Entfernung zu jedem Objekt in der Szene einzeln berechnet und die kürzeste Distanz gewählt.

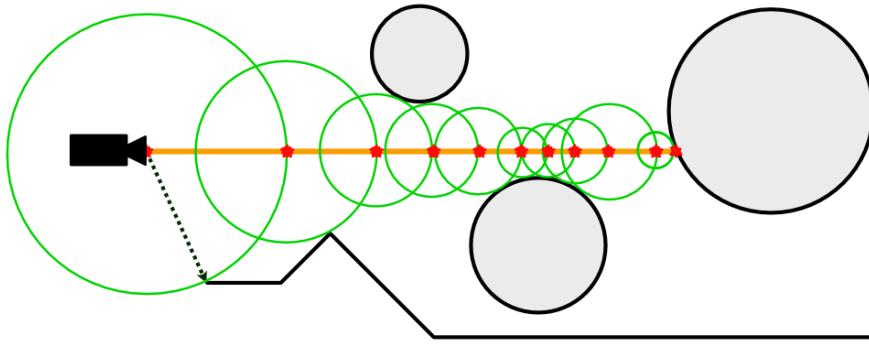


Abbildung 9: Sphere Tracing: Der Algorithmus wird so lange wiederholt, bis der Radius/Abstand zum nächstgelegenen Objekt unter einen bestimmten Grenzwert fällt.

Im ersten Schritt muss also die Distanz zur nächstgelegenen Oberfläche gefunden werden. Daraus ergibt sich ein Radius, in dem sich mit Sicherheit nichts anderes befindet. Der Strahl bewegt sich nun um die Länge des berechneten Radius in seine vorgesehene Richtung. Anschließend werden diese beiden ersten Schritte an der neuen Position wiederholt. Dies passiert so lange, bis die Entfernung gegen einen festgelegten Grenzwert läuft. Dies deutet darauf hin, dass eine Oberfläche getroffen wurde. Diese Variante von Ray Marching wird auch '*Sphere Tracing*' genannt [J. Hart, 1996]. Sphere Tracing ist, im Gegensatz zu einer konstanten Schrittweite, ein effizienter Algorithmus, um Oberflächen entlang des Strahls zu finden, da hiermit die Anzahl der benötigten Schritte deutlich reduziert werden kann.

Jede Oberfläche einer Szene kann durch eine solche Distanzfunktion dargestellt werden. Eine Kugel lässt sich also beispielsweise durch

$$\vec{d} = \text{length}(\text{center}) - \text{radius}. \quad (3.2)$$

mit ihrer Position in Weltkoordinaten und einem Radius darstellen:

Mithilfe dieser Funktionen lassen sich mehrere SDFs auf verschiedene Art und Weise kombinieren und deformieren, wodurch sich neue Formen erzeugen lassen (vgl. Abbildung 10). Dabei gibt es verschiedene Möglichkeiten, wie z.B. die Vereinigung zweier Objekte, die Schnittmenge oder auch die Differenz. Mit dem Smooth Union-Operator lassen sich beispielsweise sogenannte Blobs oder Metaballs erzeugen, indem zwischen den Oberflächen interpoliert wird, sodass diese Geometrien den Anschein erwecken, miteinander zu verschmelzen.

### 3.4.2 Volume Ray Marching

Möchte man nun aber komplexere Volumen darstellen, so gelingt dies unter Einsatz einer Volumentextur. Der Ansatz bleibt dabei der Gleiche. Es wird ein Objekt benötigt, welches als Volumen Container fungiert. Dieses kann in Form von oben beschriebenen SDFs repräsentiert sein. Innerhalb dieser Geometrie kann nun alles Mögliche dargestellt werden, da hierbei keine feste Oberfläche die Grundlage des Volumens darstellt. Es lässt sich mittels Code beispielsweise innerhalb eines Würfels eine Kugel darstellen (vgl. Abbildung 12).

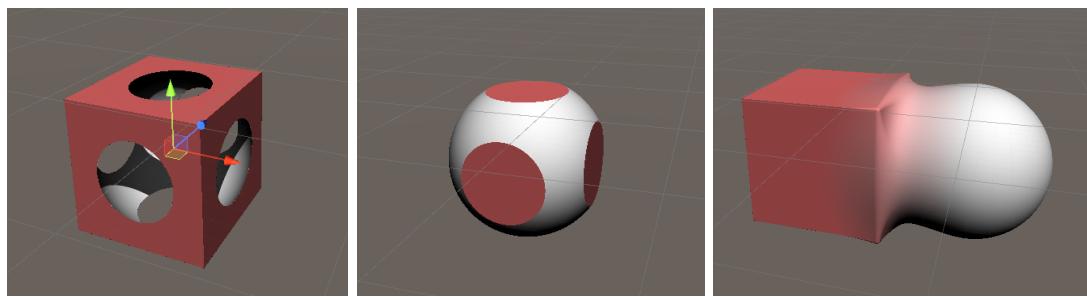


Abbildung 10: Beispiele von verschiedenen SDF-Operatoren. Von links: Subtraction, Intersection, Smooth Union.

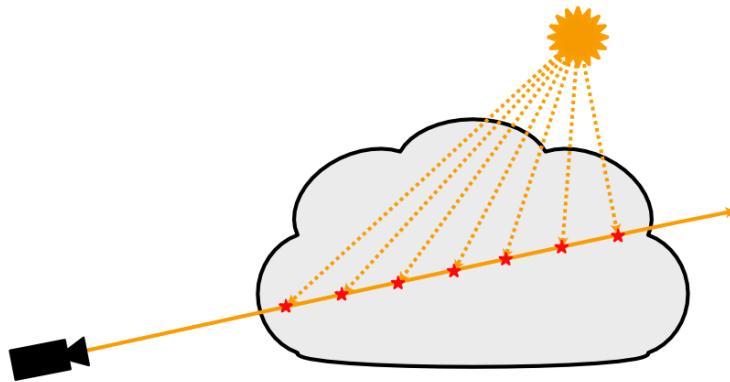


Abbildung 11: Vereinfachte Darstellung des Volume Ray Marching. Der Weg des Lichtes durch das Medium. In der Realität ist die Berechnung aufgrund von In- und Out-Scattering, Absorption oder Emissionen innerhalb des Volumens komplexer.

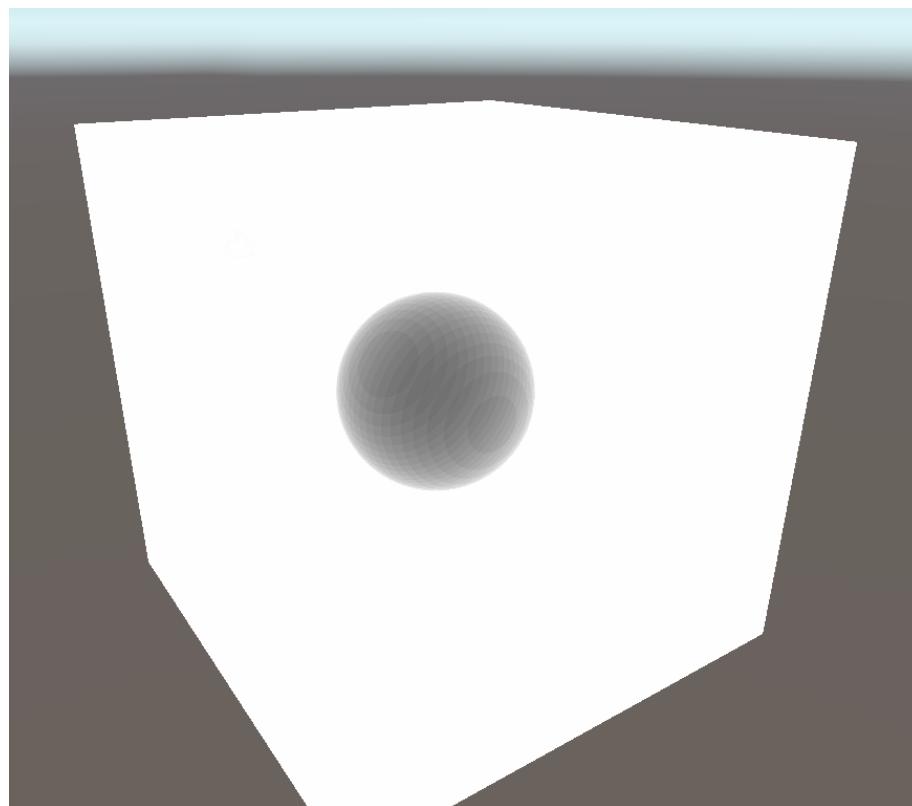


Abbildung 12: Kugel im Inneren einer Box mittels Ray Marching gerendert.

Wurde per Sphere Tracing eine Oberfläche ermittelt, so wird mit einer festgelegten Anzahl an Schritten das Innere der Geometrie durchlaufen und an jedem Schritt ein Wert berechnet, der sich aus Farbe, Dichte und Tiefe im Volumen ergibt (vgl. **Abbildung 11**). Mithilfe von Gleichung 3.3 lässt sich für jeden Strahl ein Farbwert mit Informationen aus jedem Voxel bestimmen. Voxel sind das Äquivalent zu einem Pixel im dreidimensionalen Raum und werden durch ein 3D-Gitternetz repräsentiert.

$$C_i = C_{i-1} + (1 - \alpha_{i-1}) * C(i\Delta s) \quad (3.3)$$

(Berechnung der Farbe des austretenden Lichts mit  $C_i$  = Farbe des Voxels an Stelle  $i$ ,  $\Delta s$  der Schrittweite und  $\alpha$  der Dichte des Voxels.)

## 4 Konzeption und Umsetzung

Für die Implementierung und Evaluierung der Performance der beiden beschriebenen Methoden wurde ein System mit den folgenden Spezifikationen verwendet:

Betriebssystem	Win10 64-Bit
Prozessor	Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz
RAM	16GB
Grafikkarte	NVIDIA GeForce RTX 2080 Ti
Speicher	Samsung SSD 970 EVO Plus 500GB

Tabelle 2: Spezifikationen des Computers, der im Rahmen dieser Arbeit verwendet wurde.

### 4.1 Idee

Die beiden Methoden werden für den Vergleich in der Laufzeit- und Entwicklungsumgebung Unity<sup>1</sup> entwickelt, um hinsichtlich ihrer Anwendbarkeit für Feuer und Rauch in VR-Systemen analysiert werden zu können.

Ein wichtiger Faktor bei der Darstellung von Rauch ist die Beleuchtung. Eigenschaften wie Streuungen und Reflexionen innerhalb des Mediums zu simulieren ist aufwändig zu berechnen und unter Berücksichtigung aller Faktoren eher weniger für die Echtzeitberechnung in VR-Systemen gedacht, welche deutlich höhere Anforderungen an die Performance der Anwendung haben. Der erste Ansatz basiert auf einer Idee von Mederic Chasse, Technical Art Director bei Highwire Games. Dieser beschreibt im Forum *realtimevfx.com* einen Workflow, mit dem aufwändige Berechnungen solcher Simulationen in einer VFX Software erstellt, durchsimuliert und in Texturen gebacken werden können. Die Animation wird in einem Texture Sheet gespeichert und kann somit eine realistische Darstellung bieten, während der Einsatz in einer Game Engine effizient

---

<sup>1</sup><https://unity.com>

bleibt [Chasse, 2018]. Da der Fokus dieser Arbeit nicht auf der Fluidsimulation liegt und das Setup der Simulation mithilfe von Anleitungen und Tipps artistischer Natur aus Internetforen erzeugt wird, wird an dieser Stelle nicht weiter darauf eingegangen, wie die VFX-Software den Rauch berechnet und erzeugt. Stattdessen wird in **Kapitel 4.2.1** der Workflow beschrieben und auf einige Parameter der Simulation eingegangen.

Für die Darstellung durch Ray Marching werden Volumendaten, bzw. 3D-Texturen benötigt. Diese werden aus derselben Simulation erzeugt, die auch für die Lightmaps verwendet wird. Alternativ können Volumendaten in Form von 3D-Rauschen auch mittels Code generiert werden. Es wird ein Volume Ray Marching Shader implementiert, der eine Geometrie schrittweise durchläuft und das Volumen dabei abtasten und darstellen kann. Es gibt zwei Möglichkeiten, mit denen die Volumen animiert werden können. Einerseits lässt sich das Volumen im Inneren eines solchen Containers animieren. Der Rauch kann bewegt, skaliert und rotiert werden. Andererseits können die Volumendaten statisch in Weltkoordinaten angeordnet sein und der Container animiert werden, sodass immer nur ein Ausschnitt des tatsächlichen Volumens zu sehen ist.

## 4.2 Erstellung der Texturen

### 4.2.1 Texture Sheets

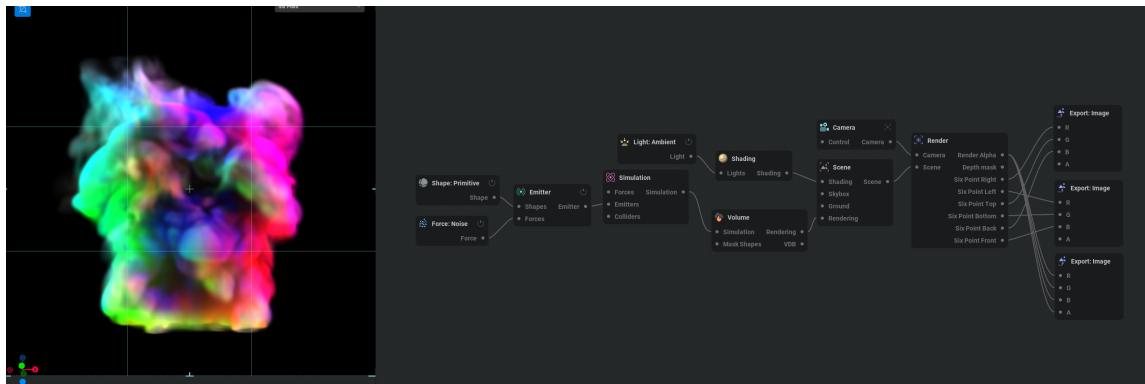


Abbildung 13: Minimales Szenensetup um eine Rauchsimulation zu erzeugen. Auf der linken Seite ist das Ergebnis des Netzwerks auf der rechten Seite zu sehen. Das Shading ergibt sich aus der integrierten 6-Punkt-Beleuchtung.

Die Texturen werden auf zwei verschiedene Arten erstellt. Für die Variante mit Parallax Occlusion Mapping werden drei verschiedene Texturen, in der von JangaFX entwickelten 3D-VFX-Grafiksoftware EmberGen<sup>2</sup>, erstellt. EmberGen wurde speziell für Feuer und Rauchsimulationen entwickelt und bietet die Möglichkeit, relativ einfach und schnell, physikalisch basierte und visuell überzeugende Rauch- und Feuersimulationen zu erstellen (vgl. **Abbildung 13**). Die Simulationen basieren auf Fluidsimulationen, wodurch ein realistisches Verhalten erzeugt, und durch eine Vielzahl an Parametern beeinflusst werden kann.

Damit der Rauch als Partikel verwendet werden kann, sollte dieser nicht wie eine komplette Rauchfahne, sonder idealerweise wie ein einzelner Rauchball geformt sein. Bei der Verwendung in einem Partikelsystem können die Partikel dadurch an einem zufälligen Ort mit zufälliger Grö-

<sup>2</sup><https://jangafx.com/software/embergen/>

ße erzeugt und eingeblendet werden. Auf diese Art wird verhindert, dass der Rauch unnatürlich aussieht. Er bleibt vor allem flexibler, was eine mögliche Interaktion des Nutzers mit dem Partikelsystem angeht. Um einen solchen Rauchball zu erzeugen, spielen einige Parameter innerhalb der VFX-Software eine wichtige Rolle. Einer dieser Faktoren ist die Dissipation, also wie schnell sich der Rauch auflöst. Dieser Wert wurde so hoch wie möglich gestellt, damit der Rauch sich nicht über eine große Fläche zieht, sondern ein kleines Volumen bleibt. Neben Einstellungen für Force-Fields, um die Simulation etwas zufälliger zu gestalten, und Parametern, die sich auf die Dichte, Geschwindigkeit und Druck des Rauchs und des Feuers beziehen, gibt es eine Vielzahl an Einstellungen für die Domäne in der die Simulation stattfindet. Die Simulation wird auf Basis von Voxeln berechnet. Es lässt sich konfigurieren, wie groß das Voxelgitter in X, Y und Z-Richtung sein soll. Dazu kann festgelegt werden, wie groß jedes Voxel für die Berechnung sein soll. Die Simulation für diese Arbeit wurde in einem 256x256x256 Gitter bei einer Voxelgröße von 10cm durchgeführt. Dies entspricht einer Gesamtanzahl von fast 16.8 Millionen Voxeln für die berechnete Simulation.

Bei dieser Simulation wird nicht nur die Beleuchtung der Oberfläche aus verschiedenen Richtungen, sondern auch die Streuung, Reflexion und Absorption des Lichts innerhalb des Volumens berechnet und simuliert. Eine Möglichkeit, die EmberGen bietet, ist das 6-Point Lighting. Mit dieser Option kann jedes einzelne Licht separat berechnet werden. Dazu werden Lichtquellen, wie in **Abbildung 14** dargestellt, aus allen sechs Richtungen zum Rauch hin angeordnet, sodass jeweils eine Seite beleuchtet wird. Die Animation des Rauches wird daraufhin wie in **Abbildung 15** in eine sich selbst nahtlos wiederholende Animation geschnitten und als einzelne Frames gerendert. Dazu muss im ersten Schritt eine Animation mit der doppelten Anzahl an Frames erzeugt werden, damit diese in der Hälfte geschnitten werden kann. Nun werden die beiden Abschnitte übereinander gelegt, sodass sich die Schnittkanten auf gegenüberliegenden Seiten befinden. So mit sind der erste und der letzte Frame nahezu identisch. Als Letztes wird ein Crossfade über beide Abschnitte gelegt, wodurch die Clips jeweils an der Schnittkante voll eingeblendet sind. Das Resultat ist, dass der Schnitt nicht mehr sichtbar ist und das Video sich unendlich wie-

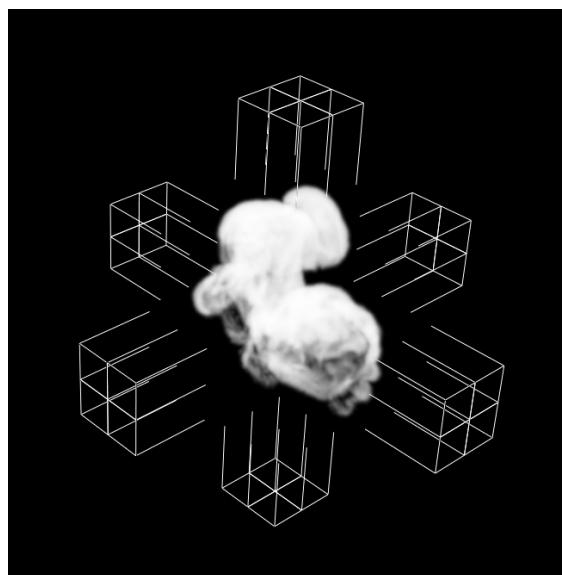


Abbildung 14: Setup der 6-Punkt-Beleuchtung einer Rauchsimulation.

derholen lässt. Die Vorteile einer sich endlos wiederholenden Animation werden in **Kapitel 5.1** noch weiter erläutert. Um die verschiedenen Texurlayers in Unity verwenden zu können, werden die Lichtrichtungen jedes einzelnen Frames in jeweils einen RGBA-Channel gepackt. So kann die Ansicht des Rauchs, in der beispielsweise nur die linke Seite beleuchtet wird, in den Roten Kanal einer Textur gespeichert werden. Dies passiert ebenfalls für die anderen Richtungen von links, unten und oben. Die Ergebnisse sind in **Abbildung 16** zu sehen.

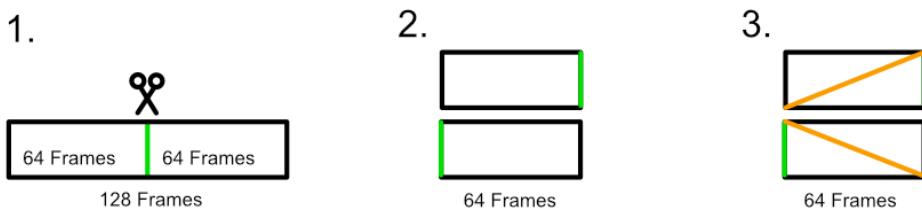


Abbildung 15: Schritte um Loop einer Rauchanimation ohne sichtbaren Schnitt zu erzeugen.



Abbildung 16: Ein einzelner Frame der Simulation in seine Farbkanäle aufgeteilt. Die Richtungen, aus denen der Rauch jeweils beleuchtet wurde, ist dabei gut zu erkennen. Von links: Roter Kanal(links), Grüner Kanal(rechts), Blauer Kanal(oben), Alpha Kanal(unten), Zusammengeführte Textur.

Die verschiedenen Farbkanäle können nun wieder in eine gemeinsame Textur zusammengeführt werden, um dann in Unity in einem Shader weiterverarbeitet zu werden. Dabei wird jeder Frame mit einer Auflösung von 256x256 Pixeln gespeichert. Bei einer Frameanzahl von  $8 \times 8 = 64$  Frames entspricht das also einer Auflösung eines Texture Sheets von 2048 x 2048 Pixeln (vgl. **Abbildung 17**).

Damit Parallax Occlusion Mapping angewendet werden kann, werden jedoch zusätzlich Informationen über die Höhe oder Tiefe benötigt. Dazu wird eine weitere Textur mit den gleichen Schritten wie zuvor erzeugt. In diese Textur wird die Tiefenkarte abgespeichert. Zudem kann in ein einem separaten Farbkanal die Alpha-Information gespeichert werden.

#### 4.2.2 Volume Textures

Für den Ray Marcher wird eine Volumentextur mit Dichtewerten benötigt. Da schrittweise durch das Volumen gelaufen und dabei überprüft wird, wie viel Licht durch dieses Volumen wandert, wird hier keine Lichtinformation in der Textur notwendig. Diese wird in Echtzeit von dem Algorithmus berechnet. Für die Volumentextur kann aus EmberGen, auf Basis derselben Simulation, eine VDB-Datei exportiert werden. VDB-Dateien können in Unity nicht verwertet werden. Diese kann jedoch in die Software Houdini von SideFX importiert werden. Von dort aus lässt sich ein einzelner Frame als Texture Sheet exportieren, wobei jeder Frame hierbei einen sogenannten Slice des Volumens darstellt. Durch diese Darstellung kann die Information in Unity als 3D-Textur

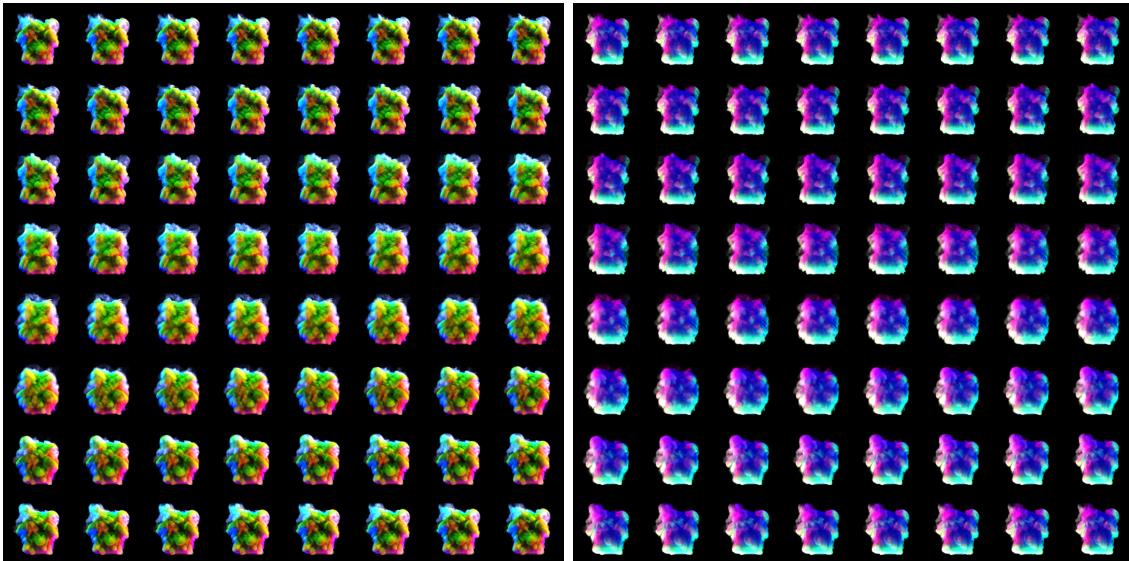


Abbildung 17: Exportierte Spritesheets der Lightmaps aus der Rauchsimulation.

verwendet werden. Hierbei wird nicht, wie bei der Animation für Parallax Occlusion Mapping, die Zeit als dritte Dimension angesehen, sondern eine weitere räumliche Achse. Die einzelnen Frames werden übereinander gelegt. Somit bildet diese Textur die Basis für eine statische Rauchwolke, welche durch Ray Marching dargestellt werden kann.

## 4.3 Shader

Für das Rendering der Partikel werden zwei Shader entwickelt. Diese basieren auf Parallax Occlusion Mapping und auf einem Ray Marching-Algorithmus durch eine Volumen Textur. Für POM wird Unitys Shader Graph verwendet. Shader Graph ist, genau wie Unitys VFX Graph, ein Node-Editor, um schnell und übersichtlich Shader zu entwickeln. Dies macht vor allem das Debugging einfacher und erlaubt das schnelle Prototyping von Ideen, da einiges an Boilerplate-Code erspart wird. Zusätzlich besteht die Möglichkeit, auch eigene Funktionen oder HLSL-Dateien in Nodes zu verpacken und somit auf visuelle Art und Weise zu entwickeln.

### 4.3.1 Parallax Occlusion Mapping

Um die einzelnen Frames aus den Texture Sheets als Animation darstellen zu können, wird ein Flipbook-Node[Technologies, 2022] verwendet. Dieser manipuliert die UV-Koordinaten mithilfe der Information über Zeilen und Spalten des Texture Sheets, sodass die Koordinaten eines jeden Frames abhängig von der Zeit ausgegeben werden. Somit lässt sich auch die Framerate anpassen, mit der die Animation abgespielt werden soll.

Diese UV-Koordinaten werden nun in einen Parallax Occlusion Mapping-Node weitergegeben, hier wird ebenfalls der Node von Unity benutzt. Dieser berechnet, basierend auf den übergebenen UV-Koordinaten des Flipbook-Players, sowie einer Heightmap und dem Blickwinkel, die anzuseigenden Texturkoordinaten. Der Algorithmus, mit dem sich das Parallax Offset berechnen lässt, wurde in **Kapitel 3.3.4** erläutert. Es lassen sich außerdem die Anzahl der Schritte mit denen die Heightmap abgetastet werden soll, sowie die Amplitude des Offsets, festlegen. Für die Implementierung des Rauches wurde die Schrittanzahl auf 150 und die Amplitude auf 1 bis

3.5 gesetzt. Diese Werte haben sich durch Ausprobieren ergeben. Eine höhere Amplitude lässt Verzerrungsartefakte auftreten.

Da die Beleuchtung des Rauches zuvor in EmberGen durchsimuliert und abgespeichert wurde, muss im Shader nun eine Logik implementiert werden, welche jeweils die richtige Beleuchtung, abhängig von der Lichtrichtung, anzeigt. Die durch den Flipbook- und POM-Node modifizierten UV-Koordinaten werden nun auf das Texture-Sheet abgebildet, und somit die Animation durchlaufen. Mit der erstellten Animation kann dann unter Einbeziehung des Main Lights (in Unity definiert als die Sonne, also ein directional light) die Richtung, aus der das Licht kommt, bestimmt werden. Directional Lights haben keine Position, lediglich eine Richtung, in die sie zeigen. Mithilfe dieser Richtung kann die korrekte Lightmap des Rauchs ausgewählt werden, oder je nach Richtung des Lichts zwischen verschiedenen Maps interpoliert werden. Im Prinzip wird also für jede Achse des Lichts abgefragt, in welche Richtung das Licht zeigt. Basierend auf dieser Info kann der entsprechende Farbchannel der Rauchtextur zurückgegeben werden.

---

**Algorithmus 1** Auswahl der richtigen Lightmap am Beispiel der X-Richtung des Lichts.

---

```

1: if lightDirection.x > 0 then
2:   return smokeLightmapT1.r
3: else
4:   return smokeLightmapT2.r
5: end if

```

---

#### 4.3.2 Raymarcher

Da Feuer und Rauch volumetrische Effekte ohne feste Oberfläche sind, lassen sich diese nicht wirklich durch Geometrie abbilden. Die Idee ist also, ein Volumen zu erzeugen und Strahlen durch dieses Volumen zu schicken, welches in festen Abständen das Volumen abtastet, um daraus Informationen zu erhalten. Hierbei wird das Licht, welches ebenfalls in das Volumen eindringt und absorbiert oder gebrochen wird, an jedem Samplepunkt berechnet.

Für den Shader wird ein Custom Function-Node für das Marching erzeugt. Der Shader arbeitet die in **Algorithmus 2** grob zusammengefassten Schritte für jeden Strahl ab. Es werden Strahlen, von der Oberfläche des Objekts in Richtung  $\vec{v} = P_{Obj} - P_{Cam}$  durch das Volumen geschickt. Dieses wird inkrementell durchlaufen. Dabei werden an jedem Punkt die Werte für die Dichte aus den Texture Slices, die das Volumen darstellen, gelesen. Zudem wird das Licht, welches den jeweiligen Punkt im Volumen erreicht, mit berechnet. Dies gelingt durch einen Ray Cast von dem jeweils abgetasteten Punkt in Richtung der Lichtquelle. Die Schritte von zuvor werden also für jeden dieser Punkte wiederholt. Basierend auf dem Lambert-beerschen Gesetz (Gleichung 3.1) wird berechnet, wie viel Licht in jedem Punkt ankommt.

Dieser Shader kann nun auf eine beliebige Geometrie angewendet werden. Da die Volumentextur einen Würfel erzeugt, kann der Einfachheit halber ebenfalls ein Würfel als Volumecontainer verwendet werden. So ist sichergestellt, dass das Volumen vollständig abgebildet werden kann.

---

**Algorithmus 2** Volume Ray Marching Algorithmus.
 

---

```

1: transparency = 1;
2: result = (0, 0, 0, 0);
3: for every sample point along viewing ray do
4:   Cast ray from sample point to light position
5:   Get distance from samplePoint to volume bounds on light ray
6:   sampleTransparency =  $\exp(-stepSize * \sigma_a)$ 
7:   transparency *= sampleTransparency
8:   result *= sampleTransparency
9: end for
10: return backgroundColor * transparency + result
  
```

---

## 4.4 Partikelsysteme

Um die Partikelsysteme umzusetzen, wird der relativ junge, von Unity entwickelte Editor *Visual Effects Graph* (kurz: VFX Graph), verwendet. VFX Graph ist ein nodesbasierter Editor, um schnell dynamische und komplexe Partikelsysteme zu erzeugen<sup>3</sup>. Im Gegensatz zum älteren Shuriken-Partikelsystem von Unity werden die Partikel hier auf der GPU simuliert, wodurch das System deutlich an Performance gewinnt, und daher mehr Partikel zeichnen kann. Shuriken nimmt die Berechnungen im Gegensatz zum VFX Graph auf der CPU vor<sup>4</sup>. Gerade für VR-Anwendungen bietet sich also dieses neue System an. VFX-Graph hat jedoch nur sehr begrenzte Möglichkeiten, was Physiksimulationen und Kollisionen der Partikel angeht. Es muss also ein System erstellt werden, welches trotz der Einschränkungen ein möglichst realistisches Verhalten der Feuer- und Rauchpartikel gewährleistet.

Das Partikelsystem setzt sich aus vier einzelnen Systemen zusammen. Dabei ist je eines für Feuer, Rauch, Funken und ein extra System für einzelne Flammen, die nach oben hin ausschlagen. Als Emitter für die Systeme dient eine Kreisfläche. Hierbei kann der Radius verändert werden, um das Ausmaß des Brandes zu bestimmen. Außerdem wird auf Basis eines 3D-Perlin Noise-Nodes ein Wind simuliert, der die Richtungen der Partikel ein wenig variieren lässt.

Das Partikelsystem, welches Funken emittiert, besitzt kein besonderes Shading. Diese sind lediglich Farbgradienten, die jedoch mit einer zufälligen Beschleunigung in eine ebenfalls zufällige Richtung initialisiert werden.

Feuer und Rauch werden jeweils mit POM oder Volume Ray Marching Shadern emittiert. Diese bekommen – neben den üblichen Eigenschaften, wie Position, Beschleunigung, Lebensdauer, Farbe und Größe – einige Parameter übergeben, mit denen die Shader jeweils gesteuert werden können (vgl. Abbildung 18). Auf diese Art und Weise lassen sich die Parameter auch animieren. Es gibt einen 'Total Time VFX'-Node, der einen Timer beinhaltet, der für die Zeit der Simulation des Systems läuft. Basierend auf diesem Node wird beispielsweise das Perlin-Noise, also die Windrichtung, animiert.

Die Feuerpartikel werden auf Höhe des Emitters erzeugt und erhalten nicht viel Auftrieb. Ist die Anzahl hoch genug, so lässt sich ein einigermaßen realistisches Feuer erzeugen. Die Lebensdauer

---

<sup>3</sup><https://unity.com/de/visual-effect-graph>

<sup>4</sup><https://docs.unity3d.com/Manual/ChoosingYourParticleSystem.html>

ist dabei relativ gering eingestellt um zu verhindern, dass die Wiederholungen der animation auffallen. Bei Feuer geschieht dies aufgrund der Helligkeit der Texturen schenller, als bei dunklem Rauch. Der Rauch verhält sich ein wenig anders. Dieser bleibt normalerweise nicht am Boden, sondern steigt auf und wird meist erst über dem Feuer sichtbar. Die Lebensdauer der Rauchpartikel ist dabei länger eingestellt, da der Loop hier weniger auffällt. Damit die Partikelsysteme plausibler aussehen und die Partikel nicht einfach existieren und plötzlich verschwinden, werden diese bei Entstehung und Vernichtung Ein- und Ausgeblendet.

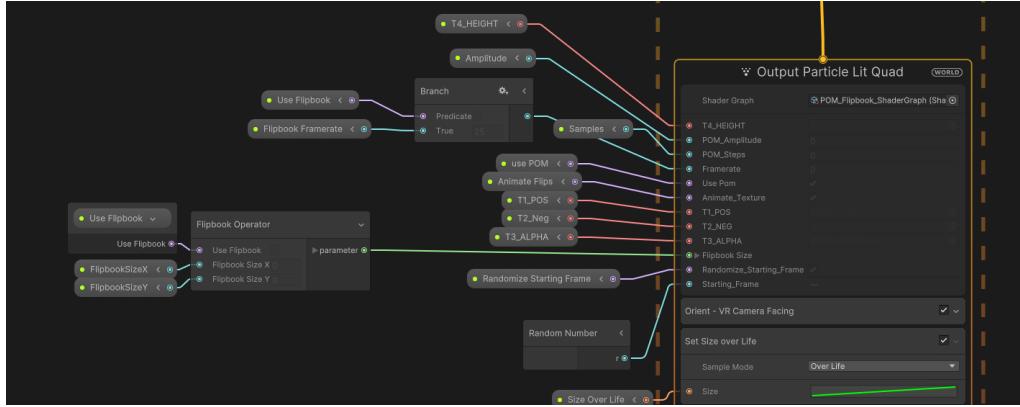


Abbildung 18: Ausschnitt des Visual Effects Graphs am Beispiel des POM-basierten Systems. Der dargestellte Ausschnitt ist für die Steuerung des Shaders zuständig.

## 5 Evaluierung

Beide Methoden haben ihre Vor- und Nachteile. Die realistische Darstellung von voluminösen Materialien verlangt eine besondere Herangehensweise. Die Darstellung der Volumen lässt sich – im Gegensatz zu undurchsichtigen, festen Oberflächen – mit herkömmlichen Texture-Mapping-Methoden in einer VR-Umgebung nicht überzeugend darstellen. Die Implementierung der Systeme ist aufgrund der begrenzten Zeit im Rahmen dieser Arbeit nur prototypisch mithilfe der zur Verfügung stehenden Funktionen aus Unity implementiert, und daher nicht bestmöglich optimiert. Einige Vor- und Nachteile, sowie das Potential für den Nutzen der jeweiligen Methoden, lassen sich dennoch aus dem Entwurf ableiten und werden in den weiteren Abschnitten beschrieben.

### 5.1 Parallax Occlusion Mapping

POM ist eine komplexere Variante des Normalmappings. Dementsprechend ist auch diese Technik eher für feste Oberflächen und weniger für den Einsatz bei der Simulation von Volumen gedacht. Um aber die Effizienz der Billboard-basierten Partikelsysteme nutzen zu können, ist das Parallax Occlusion Mapping aber eine interessante Möglichkeit, die es wert ist, getestet zu werden. Vorallem für die Anwendung bei dichtem, schwarzen Rauch mit geringer Transparenz, bei dem die äußere Schicht bei Betrachtung aus weiterer Entfernung beinahe wie eine solide Oberfläche erscheinen könnte.

In Abbildung 19 ist ein Vergleich der Variante von POM mit einem einzelnen Sprite, sowie einer Animation, basierend auf einem Texture Sheet, zu sehen. Auf der linken Seite befindet

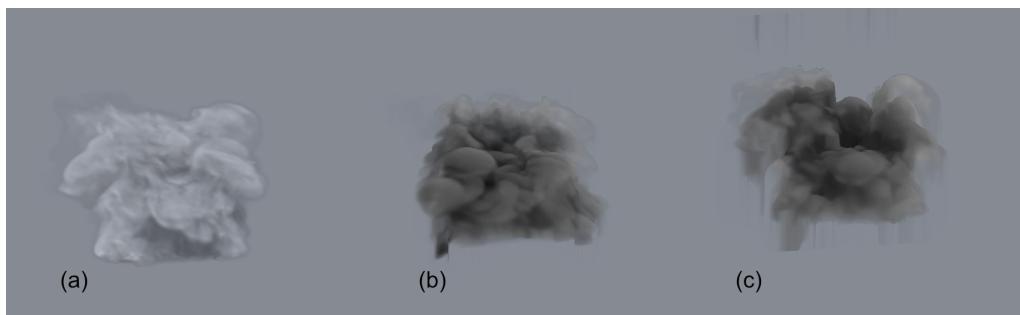


Abbildung 19: Vergleich verschiedener Verfahren, aus flacherem Blickwinkel betrachtet. (a) Normal Mapping, (b) 6-Punkt-beleuchtetes Parallax Occlusion Mapping, einzelner Frame der Simulation. (c) 6-Punkt-beleuchtetes Parallax Occlusion Mapping, Texture Sheet-Animation.

sich ein Frame der Simulation ohne Parallax Offsetting, lediglich mit einer Normal Map. Dieser Vergleich macht den Vorteil der 6-Punkt-Beleuchtung und des Parallax Occlusion Mappings gegenüber herkömmlichen Mappingverfahren deutlich. Bei der Betrachtung in VR entsteht ein realistischerer Eindruck von Tiefe, da sich blickwinkelabhängig verschiedene Teile der Textur gegenseitig verdecken. Gerade wenn sich beim Betrachten um die Textur herum bewegt wird, verstärkt sich dieser Effekt. Dies bestätigt die Vermutung, dass sich POM für VR deutlich mehr eignet als herkömmliches Normal Mapping. Jedoch gilt dies nur für das Standbild des Rauches. Auf der rechten Seite sieht man deutlich, wie sich die benachbarten Frames bei der Animation des Rauches mit in den sichtbaren Bereich hineingezerrt werden. Ein weiteres Problem der Texture Sheet-basierten Animation sind die Randflächen des Rauches. Dadurch, dass die Höhe an der Umrandung des Rauches in der Textur unmittelbar abfällt, werden einige Pixel, die an einer solchen Schnittkante liegen, stark verzogen. Selbst bei geringem Pixeloffset, bei dem sich der Parallax-Effekt noch kaum bemerkbar macht. Auch dies ist in **Abbildung 19** erkennbar. Da sich der Rauch mit jedem Frame verändert, springen die Artefakte entlang der Randflächen. Der Effekt tritt stärker auf, wenn zunächst die Animation durchlaufen und daraufhin das Pixeloffset berechnet wird. Die Artefakte lassen sich reduzieren, indem die Reihenfolge der UV-Manipulationen umgekehrt wird. Die Implementierung des POM Nodes von Unity verhindert damit jedoch die Animation der Heightmap, was die Texture Sheet-Animation – in Kombination mit Parallax Occlusion Mapping – in der Art und Weise, wie sie hier implementiert ist, unbrauchbar macht.

In Anbetracht der Performance läuft die Anwendung auf dem Testsystem in einer VR-Umgebung, stabil mit 90FPS. Trotz der erhöhten Komplexität des Algorithmus gegenüber klassischeren Mappingverfahren hält sich der Rechenaufwand in Grenzen. Hier gibt es also keine Bedenken hinsichtlich der Shaderlaufzeit. Auch die aufwändige Berechnung der Fluidsimulationen wird hier im Voraus erledigt. Somit ist es möglich, realistisch aussehende Partikelsysteme unter geringem Rechenaufwand in der Echtzeitanwendung zu erzeugen. Ein weiterer Vorteil, der sich aus der Natur des schwarzen Rauches bei Bränden ergibt, ist die Tatsache, dass die genaue Berechnung von Selbstschattierungen hierbei nicht so essentiell ist wie bei weißem Rauch. Dies ist zwar nicht direkt ein Vorteil von POM, gleicht aber einen Nachteil aus, den es in der Anwendung in einem Partikelsystem gibt: Die Texturen auf den Billboards sind in der Lage, sich individuell selbst zu schattieren. Eine gegenseitige Schattierung ist jedoch ohne weiteres nicht möglich (vgl. **Abbildung 20**). In Bezug auf Schattierung im Allgemeinen gibt es das Problem, dass Billboard-

basierte Partikelsysteme eigentlich nur flache Sprites sind und deshalb nicht in der Lage sind, realistische Schatten auf die Umgebung werfen zu können.

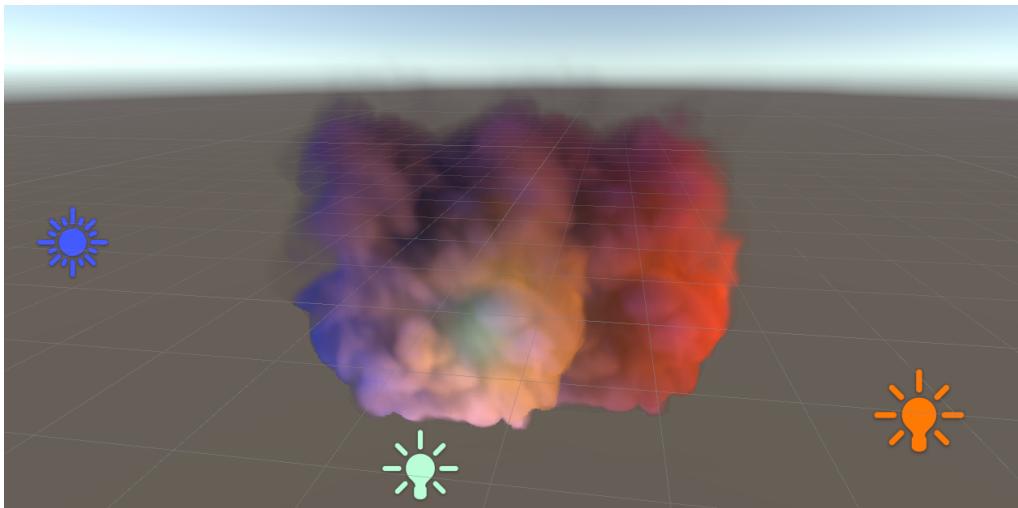


Abbildung 20: Kein gegenseitiger Schattenwurf bei POM-basierten Billboards.

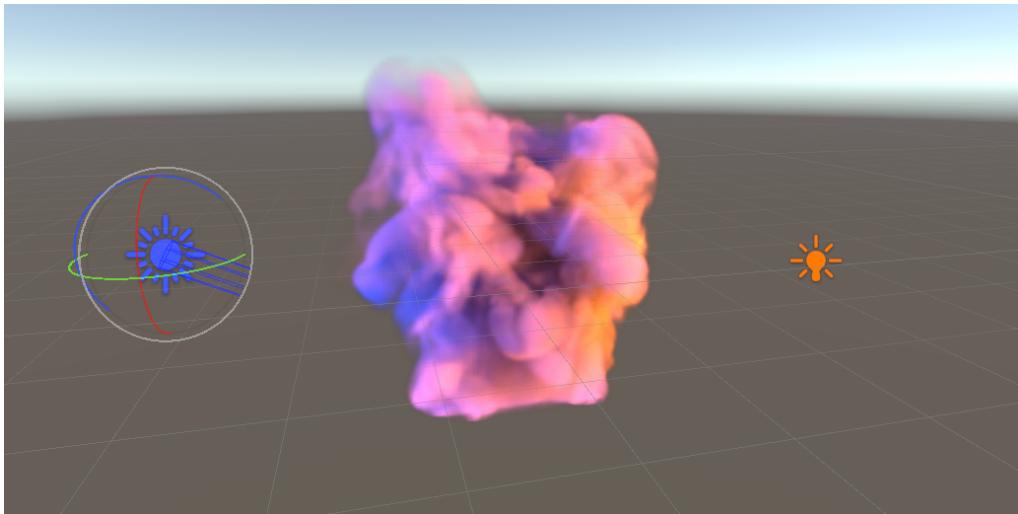


Abbildung 21: Dynamische Beleuchtung des Rauchs

Ein weiterer Vorteil, der sich aus dieser Methode ergibt, sind die Texture Sheets. In eine einzige Textur lassen sich 4 verschiedene Informationen in die RGBA-Kanäle speichern. Neben den verschiedenen Lightmaps aus 6 Richtungen können beispielsweise noch Farbinformationen oder Emission- und Heatmaps gespeichert werden. Mit weiteren Infos wie einer Heatmap lässt sich ein Übergang, bzw. eine Mischung von Feuer und Rauch erzeugen. Dies ist ein Phänomen, welches sich bei größeren Bränden oder Explosionen beobachten lässt (vgl. Abbildung 2). Die Anzahl der Lightmaps lässt sich auch reduzieren, indem die Lichtrichtungen von vorne und von hinten im Shader approximiert werden. Es gibt viele Möglichkeiten, die Flexibilität von Texturen auszunutzen und dabei die Menge an benötigten Daten und Berechnungen gering zu halten, sodass der flüssige Einsatz in VR gegeben bleibt. Weiterhin lassen sich auf Kosten weiterer Texture Sheets verschiedene Simulationen erstellen, sodass größere Varianz in das Aussehen des Partikelsystems eingebracht werden kann. Durch die unendliche Wiederholbarkeit kann die Animation ab einem zufällig gewählten Frame starten, wodurch schon mit einer einzigen Animation mit genug Frames kaum auffällt, dass das ganze System auf lediglich einer einzigen Sequenz

aufbaut.

Neben den ganzen Vorteilen, die diese Variante mit sich bringt, gibt es auch negative Punkte. Bei zu geringer Auflösung sehen die Renderings schnell verpixelt aus, je näher diese betrachtet werden. Dies ist ein großer Nachteil der Texture Sheets. Wird die Auflösung jedes Frames verdoppelt, also in diesem Fall von 256x256 Pixel auf 512x512 Pixel, so erhöht sich die Auflösung der gesamten Textur um den Faktor vier für jeden der RGBA-Kanäle. Selbst dann sind die einzelnen Frames noch relativ gering aufgelöst und nicht für die Betrachtung aus der Nähe gedacht. Je detaillierter die Simulation, auf denen die Texture Sheets basieren, desto höher sollte die Auflösung sein, um auch alle diese Details abbilden zu können. Die visuellen Artefakte, wie beispielsweise das Einschneiden der benachbarten Frames aus dem Texture Sheet aus einem flachen Betrachtungswinkel (vgl. Abbildung 22), schränken die Einsatzmöglichkeiten für die Anwendung in VR stark ein. Die Verzerrung am Rand, welche durch eine starke Kante in der Heightmap hervorgerufen wird, macht das Parallax Occlusion Mapping ebenfalls für die Darstellung von voluminösen Medien ungeeignet. Zudem ist der Parallax Effekt bei Billboards, die immer von vorne betrachtet werden, so schwach, dass sich der Aufwand der Berechnungen nicht wirklich lohnt. Bei der Darstellung des Feuers kommt der Faktor hinzu, dass Billboards nicht als Lichtquelle fungieren, was dem Realismus der Simulation schadet.

Abschließend lässt sich zu dem Rendering von Feuer und Rauch auf Basis von Parallax Occlusion Mapping sagen, dass sich diese Variante, so wie sie für diese Arbeit implementiert ist, nicht immer eignet. Die visuellen Artefakte erzeugen Störungen der Immersion. Jedoch wird das Parallax Offset in der Stereoansicht korrekt dargestellt, wodurch dieser Ansatz vielversprechend bleibt. Es ist wahrscheinlich, dass es für die genannten Probleme elegante Lösungen gibt, die in Zukunft an anderer Stelle betrachtet werden könnten.

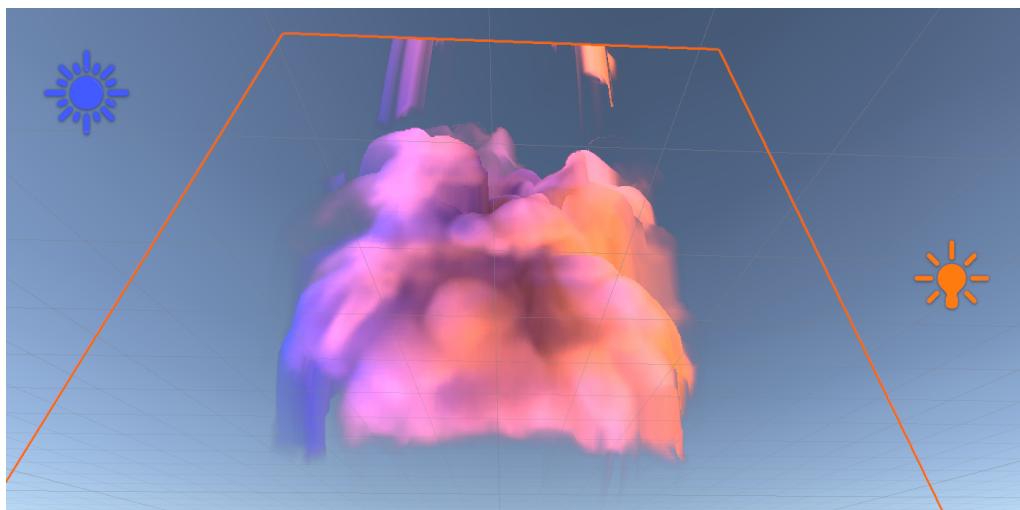


Abbildung 22: Verzerrungsartefakte bei der Anwendung von Parallax Occlusion Mapping mit zu starkem Offset. Die benachbarten Frames des Texture Sheets werden ebenfalls mit verzerrt, wodurch Fragmente davon, bei flachem Betrachtungswinkel, mit in das Billboard einlaufen.

## 5.2 Ray Marching

Aufgrund der massiven Einschränkungen in der Qualität des Exports aus Houdini wurde (für die bessere Betrachtung in VR) ein Set Volume Slices aus dem Github-Repository von Dylan Meville

verwendet<sup>5</sup>. Der Ansatz des Volume Renderings auf Basis von Ray Marching eignet sich besonders gut für die Darstellung von Effekten wie Feuer und Rauch. Die Tatsache, dass das Volumen nicht nur vorgetäuscht, sondern tatsächlich dargestellt werden kann, macht das Rendering sehr realistisch. Die Volumen werden durchlaufen und dabei wird schrittweise die Farbe, die Dichte, sowie die Lichtintensität berechnet. Daraus ergibt sich ein qualitativ hochwertiges Shading der Rauchwolken. Gerade für die Betrachtung in VR ist dieser Ansatz besonders vielversprechend. Die Volumen sehen, aus allen Richtungen betrachtet, realistisch aus und können auf dynamische Beleuchtung reagieren. Zudem können Volumen das durchlaufende Licht nicht nur absorbieren und reflektieren, sondern auch selbst emittieren. Durch diese Eigenschaft kann das emittierte Licht die Umgebung nicht nur schattieren, sondern auch beleuchten.

Verfolgt man die Lichtstrahlen durch die Volumen weiter, so können mit geringem Mehraufwand auch realistische Schattierungen der Umgebung berechnet werden. Wurde ein Volumen, bzw. eine Szene, mittels Ray Marching abgelaufen, so können, ebenfalls mit geringem Aufwand, weitere Effekte, wie Ambient Occlusion oder Reflektionen berechnet werden. All dies sind Faktoren, die zu einem realistischeren Rendering beitragen.

Ebenso wie Parallax Occlusion Mapping, werden die Berechnungen hier für jeden Pixel vorgenommen. Das heißt, dass aufgrund der Natur des Ray Marching Algorithmus, ohne Probleme für jedes Auge eine korrekte Darstellung geliefert werden kann. Dabei muss jedoch gerade in VR-Umgebungen die Performance beobachtet werden. In der Testszene lief die Betrachtung des ray marched Rauches mit einer stabilen Frame Rate oberhalb der 90FPS-Grenze. Somit eignet sich auch diese Methode für die Anwendung in VR. Mit steigender Komplexität der Berechnungen, beispielsweise durch die zusätzlichen Effekte oder komplexere Szenen, könnte die Performance jedoch darunter leiden.

Ein Nachteil der vorgeschlagenen Implementierung des Volume Ray Marching liegt bei den notwendigen 3D-Texturen. Die Texturen auf Basis der Fluidsimulationen sind statisch und lassen sich nicht realistisch animieren. Lediglich die Texturen, basierend auf dem generierten Rauschen, können animiert werden, haben aber den Nachteil, dass diese weniger realistisch aussehen. Diese Texturen würden sich eher anbieten, um weit entfernte Wolken oder Nebel zu erzeugen. Der dichte, schwarze Rauch, der bei einem großen Feuer erzeugt wird, lässt sich nicht überzeugend durch solche Texturen darstellen.

Letztendlich ist der Volume Ray Marching-Ansatz jedoch auch sehr vielversprechend und kann in einer Simulation in VR gut verwendet werden. Die Methode bietet einige Möglichkeiten der Optimierung und erzeugt ein plausibles Bild für den Betrachter. Für die Anwendung in einem Partikelsystem für Feuer und Rauch muss jedoch noch eine Möglichkeit gefunden werden, den Rauch zu animieren oder realitätsnah geformt werden zu können.

---

<sup>5</sup><https://github.com/DMeville/DMVolumeRendering/blob/main/DMVolumeRendering/Assets/VolumeRubberCube.exr>, abgerufen am 28.07.2022

# 6 Schluss

## 6.1 Fazit

Das Ziel dieser Bachelorarbeit ist es, zwei vielversprechende Methoden für das Stereorendering von Feuer und Rauch testweise zu implementieren, zu vergleichen und die Eignung für den möglichen Einsatz in VR-Systemen zu beurteilen. Nach Evaluierung der Ergebnisse von Parallax Occlusion Mapping und Ray Marching stellt sich heraus, dass beide Algorithmen potentiell für das Rendering von Rauch geeignet sein könnten.

Die Darstellung der POM-basierten Billboards, so wie sie im Rahmen dieser Arbeit in Kombination mit den Texture Sheets für die Betrachtung in VR verwendet wurde, bringt jedoch auch einige Probleme, wie die beschriebenen Artefakte und Verzerrungen mit sich. Parallax Occlusion Mapping hat den großen Vorteil, dass die Offsets und Verdeckungen basierend auf dem Blickwinkel berechnet werden. Dies ist eine Stärke, die für den Einsatz in Virtual Reality mit HMDs spricht. Jedoch ist aus den Erkenntnissen dieser Arbeit der Volume Rendering-Ansatz vorzuziehen. Durch das Rendering eines tatsächlichen Volumens sieht der Rauch in virtueller Realität realistischer aus, als durch eine Illusion auf einem zweidimensionalen Sprite. Es gibt viele kleine Tricks, um den Ray Marching-Algorithmus zu optimieren und die Anzahl der Samples zu reduzieren, die im Rahmen dieser Arbeit jedoch nicht alle implementiert wurden. Die vielen Vorteile des Volume Ray Marching machen dieses jedoch zu einer vielversprechenden Methode.

## 6.2 Limitationen

Die Erkenntnisse dieser Arbeit beziehen sich auf sehr grundlegende Implementierungen der beiden Varianten als Proof of Concept. Dies hängt zum einen mit Unitys Universal Rendering Pipeline zusammen, welche einerseits vorteilhaft für VR-Anwendungen ist, es andererseits aber beispielsweise schwieriger macht, auf die Lichter in der Szene zuzugreifen. Zudem ist die Dokumentation der Software, gerade was die Programmierung von Shadern angeht, teilweise nicht hilfreich. Fehlermeldungen sind nicht wirklich verständlich, und es gibt keine weitergehenden Informationen. Es musste viel auf Ideen und Meinungen aus Foren oder Videos zurückgegriffen werden, um beide Methoden im zeitlichen Rahmen implementieren zu können. Für jemanden, der nicht besonders geübt mit der Programmierung von Shadern in Unity ist, wird die Entwicklung stark verlangsamt. Weiterhin wurde für die Erstellung der Light- und Densitymaps zunächst auf eine andere Software, Houdini von SideFX zurückgegriffen. Houdini bietet eine Non-Commercial-Version an, die einige Limitationen mit sich bringt. Darunter beispielsweise eine begrenzte Auflösung der Texturen, teilweise fehlende Informationen beim Rendern der Bildsequenzen und Wasserzeichen in den Texture Sheets. Die Erstellung und Aufbereitung der Texturen hat viel Zeit in Anspruch genommen, wobei das Ergebnis daraus am Ende aus qualitativen Gründen nicht nutzbar ist, sodass die Software gewechselt werden musste, um diesen Ansatz weiter verfolgen zu können. Mit mehr Zeit und Ressourcen lassen sich hier jedoch brauchbare Ergebnisse erzeugen, die letztendlich die Qualität des Partikelsystems erhöhen könnten.

### 6.3 Ausblick

Diese Arbeit zeigt auf, dass die beiden Methoden für das Gebiet der Feuer und Rauchsimulation für virtuelle Umgebungen Potential haben, allerdings auch noch weiterentwickelt werden müssen, um einsatzfähig zu werden. Die Simulation von Phänomenen wie Feuer und Rauch ist ein breit gefächertes und komplexes Feld. Es gibt unzählige Optimierungsmöglichkeiten, was es schwerer macht, einen Fokus auf bestimmte Dinge zu setzen. Viele Themen, die ein Feuer realistischer machen, wie beispielsweise die korrekte Beleuchtung der Umgebung oder das korrekte Verhalten der Partikelsysteme, konnten in dieser Arbeit aufgrund der Zeit und ihrer Komplexität nicht angegangen werden und könnten für weitere Forschungsansätze interessant sein. Zudem könnte mehr Variation und damit Realismus durch verschiedene Animationen erzeugt werden. Gerade im Bereich der prozeduralen Texturgenerierung könnte es interessant sein, eine Art 4D-Textur zu entwickeln, die eine Animation im dreidimensionalen Raum ermöglicht. Die Möglichkeiten des Ray Marchings sind nur durch die Rechenleistung begrenzt. Mit immer besser werdender Hardware werden auch die potentiellen Einsatzmöglichkeiten des Ray Marchings immer mehr.

## Literatur

- Blinn, James (1978). „Simulation of wrinkled surfaces“. In: *Proceedings of the 5th annual conference on Computer graphics and interactive techniques - SIGGRAPH '78*. ACM Press, S. 286–292. DOI: 10.1145/800248.507101.
- Blinn, James und Martin E. Newell (Okt. 1976). „Texture and Reflection in Computer Generated Images“. In: *Commun. ACM* 19.10, S. 542–547. ISSN: 0001-0782. DOI: 10.1145/360349.360353.
- Brawley, Zoe und Natalya Tatarchuk (2004). „Parallax occlusion mapping: Self-shadowing, perspective-correct bump mapping using reverse height map tracing“. In: *ShaderX<sup>3</sup>: Advanced Rendering with DirectX and OpenGL*. Hrsg. von W. Engel. Charles River Media, S. 135–154.
- Catmull, Edwin (1974). „Computer display of curved surfaces“. Diss., S. 35–41. DOI: 10.1145/280811.280920.
- Chasse, Mederic (2018). *Smoke Lighting and texture re-usability in Skull & Bones*. URL: <https://realtimevfx.com/t/smoke-lighting-and-texture-re-usability-in-skull-bones/5339/24> (besucht am 20.07.2022).
- Cohen, Jonathan, Marc Olano und Dinesh Manocha (1998). „Appearance-Preserving Simplification“. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '98. Association for Computing Machinery, S. 115–122. DOI: 10.1145/280814.280832.
- Cook, Robert L. (1984). „Shade Trees“. In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '84. Association for Computing Machinery, S. 223–231. DOI: 10.1145/800031.808602.
- Cook, Robert L., Loren Carpenter und Edwin Catmull (1987). „The Reyes Image Rendering Architecture“. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. Association for Computing Machinery, S. 95–102. DOI: 10.1145/37401.37414.
- Dachsbaecher, Carsten und Natalya Tatarchuk (2007). *Prism Parallax Occlusion Mapping with Accurate Silhouette Generation*. URL: <https://hal.inria.fr/inria-00606806> (besucht am 30.06.2022).
- El Jamiy, Fatima und Ronald Marsh (Apr. 2019). „A Survey on Depth Perception in Head Mounted Displays: Distance Estimation in Virtual Reality, Augmented Reality and Mixed Reality“. In: *IET Image Processing* 13. DOI: 10.1049/iet-ipr.2018.5920.
- Gateau, Samuel und Steve Nash (20. Sep. 2010). *Implementing Stereoscopic 3D in Your Applications*. NVIDIA. URL:

- [https://www.nvidia.com/content/GTC-2010/pdfs/2010\\_GTC2010.pdf](https://www.nvidia.com/content/GTC-2010/pdfs/2010_GTC2010.pdf) (besucht am 30.06.2022).
- Hart, J. C., D. J. Sandin und L. H. Kauffman (1989). „Ray Tracing Deterministic 3-D Fractals“. In: *SIGGRAPH Comput. Graph.* 23, S. 289–296. DOI: 10.1145/74334.74363.
- Hart, John (1996). „Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces“. In: *The Visual Computer* 12. DOI: 10.1007/s003710050084.
- HP Development Company, L.P (2022). *HP Reverb G2 Virtual Reality Headset Specifications*. URL: <https://support.hp.com/de-de/document/c06938191#AbT2> (besucht am 30.06.2022).
- Kaneko, Tomomichi, Toshiyuki Takahei, Masahiko Inami, Naoki Kawakami, Yasuyuki Yanagida, Taro Maeda und Susumu Tachi (Jan. 2001). „Detailed shape representation with parallax mapping“. In: *In Proceedings of the ICAT 2001*.
- Kufner, Robert (2017). „Quasi volumetrisches Rendering einer gridbasierten 2D Smokesimulation unter Nutzung von Z-buffer und Parallax Mapping“. Masterarb. Institut für Informatik. Ludwig-Maximilians-Universität München.
- Mayerhöfer, Thomas G, Susanne Pahlow und Jürgen Popp (2020). „The Bouguer-Beer-Lambert law: Shining light on the obscure“. In: *ChemPhysChem* 21.18, S. 2029–2046.
- McGrattan, Kevin B und Glenn P Forney (2004). *Fire Dynamics Simulator User's Guide (Sixth Edition)*. Techn. Ber. Version 6. DOI: 10.6028/NIST.SP.1019.
- Reeves, William (1983). „Particle systems—a technique for modeling a class of fuzzy objects“. In: *Proceedings of the 10th annual conference on Computer graphics and interactive techniques - SIGGRAPH '83*. ACM Press, S. 359–375. DOI: 10.1145/800059.801167.
- Schlager, Bettina (2017). „Building a Virtual Reality Fire Training with Unity and HTC Vive“. In: *Proceedings of Central European Seminar on Computer Graphics for students (CESCG 2017)*. URL: <https://www.vrvis.at/publications/PB-VRVis-2017-008>.
- Schmidt, C.W. und S.A. Symes (2011). *The Analysis of Burned Human Remains*. ATLAS OF SURGICAL PATHOLOGY. Elsevier Science. ISBN: 9780080559285. URL: <https://books.google.de/books?id=Q7Pb2wXV2woC>.
- Schuemie, Martijn J, Peter Van Der Straaten, Merel Krijn und Charles APG Van Der Mast (2001). „Research on presence in virtual reality: A survey“. In: *CyberPsychology & Behavior* 4.2, S. 183–201.
- Sutherland, Ivan E. (1965). „The Ultimate Display“. In: *Proceedings of the IFIP Congress*, S. 506–508.
- (1968). „A head-mounted three dimensional display“. In: *AFIPS Conference Proceedings*. Bd. 33. ACM, S. 295–302. DOI: 10.1145/280811.281016.

- Tatarchuk, Natalya (2006). „Practical Parallax Occlusion Mapping with Approximate Soft Shadows for Detailed Surface Rendering“. In: *ACM SIGGRAPH 2006 Courses*. SIGGRAPH '06. Association for Computing Machinery, S. 81–112. DOI: 10.1145/1185657.1185830.
- Tauer, H. (2010). *Stereo-3D: Grundlagen, Technik und Bildgestaltung*. Fachverlag Schiele & Schoen, S. 20–70. ISBN: 9783794907915. URL: <https://books.google.de/books?id=mfJ4HjD6CwEC> (besucht am 26.06.2022).
- Technologies, Unity (22. Juli 2022). *Unity Documentation*. URL: <https://docs.unity3d.com/Manual/index.html>.
- Vries, Joey De (o.D.[a]). *Parallax Mapping*. [Online; zugegriffen am 28.Juli, 2022]. URL: [https://learnopengl.com/img/advanced-lighting/parallax\\_mapping\\_depth.png](https://learnopengl.com/img/advanced-lighting/parallax_mapping_depth.png).
- (o.D.[b]). *Parallax Occlusion Mapping*. [Online; zugegriffen am 28.Juli, 2022]. URL: [https://learnopengl.com/img/advanced-lighting/parallax\\_mapping\\_parallax\\_occlusion\\_mapping\\_diagram.png](https://learnopengl.com/img/advanced-lighting/parallax_mapping_parallax_occlusion_mapping_diagram.png).
- Wald, Ingo, Andreas Dietrich, Carsten Benthin, Alexander Efremov, Tim Dahmen, Johannes Gunther, Vlastimil Havran, Hans-peter Seidel und Philipp Slusallek (2006). „Applying Ray Tracing for Virtual Reality and Industrial Design“. In: *2006 IEEE Symposium on Interactive Ray Tracing*, S. 177–185. DOI: 10.1109/RT.2006.280229.
- Welsh, Terry (2004). „Parallax mapping with offset limiting: A perpixel approximation of uneven surfaces“. In: *Infiscape Corporation*.
- Williams-Bell, F. M., B. Kapralos, A. Hogue, B. M. Murphy und E. J. Weckman (Mai 2014). „Using Serious Games and Virtual Simulation for Training in the Fire Service: A Review“. In: *Fire Technology* 51.3, S. 553–584. DOI: 10.1007/s10694-014-0398-1.
- Zhang, Tianli (2020). „FOLAR: A FOggy-LAser Rendering Method for Interaction in Virtual Reality“. Masterarb. Computer, Information Sciences, KTH School of Electrical Engineering und Computer Science. URL: <http://www.diva-portal.org/smash/get/diva2:1420642/FULLTEXT01.pdf> (besucht am 30.06.2021).

## Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorgelegte Abschlussarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

---

Ort, Datum

---

Rechtsverbindliche Unterschrift

TH Köln  
Gustav-Heinemann-Ufer 54  
50968 Köln  
[www.th-koeln.de](http://www.th-koeln.de)

Technology  
Arts Sciences  
**TH Köln**